Oral History of Dan Ingalls

Interviewed by:
Hansen Hsu

Recorded April 10, 2017
Mountain View, CA

CHM Reference number: X8167.2017

Hsu: Today is April 10th, 2017. I am Hansen Hsu and we're here today with Dan Ingalls and so we'll start at the beginning with our questions. Where were you born and what year?

Ingalls: I was born in Washington, D.C., in 1944. I was there because my father was stationed at the Pentagon.

Hsu: Oh. So was he working with the Department of Defense?

Ingalls: Well, he was with the OSS. He was a Sanskrit Scholar and as a result knew all sorts of languages and they had convened a crew of people who knew a lot of languages to help with decoding.

Hsu: Oh, that's fascinating.

Ingalls: [He enjoyed the situation] because they had collected [many of his brightest linguistically-oriented friends] from Harvard and other universities and brought them all together. They had a good time together when they weren't on the job and the decoding challenges were all interesting.

Hsu: And did he continue that line of work after the war?

Ingalls: Well, Sanskrit [studies] yes, throughout his life.

Hsu: But in a more academic setting rather than a defense–

Ingalls: No. Sanskrit's not even a live language. No. He was a professor at Harvard. There aren't too many places [with such a position available].

Hsu: Right. And your mother, what was her occupation?

Ingalls: She was pretty much a housewife at the time.

Hsu: What are your parents' ethnic backgrounds?

Ingalls: Both just Anglo-Saxon <inaudible>.

Hsu: Do you have any siblings?

Ingalls: Yes. I have two sisters. They're both older than me and one is a painter and she lives in [North Dartmouth, Massachusetts], and she [has also taught] art at the Wheeler School [in Providence, Rhode Island] for a long time. My other sister, Rachel, lives in London and she's a writer; she [writes mostly novellas, often with a supernatural slant].

Hsu: Did your family have any religious or political leanings or affiliations growing up?

Ingalls: Not really, no. I mean I think mostly my father's work was deeply connected with his scholarly pursuits and the family before that was involved in running a hotel in Virginia. That's what my grandfather did.

Hsu: Oh, interesting. Did your grandfather's family have deep roots in Virginia?

Ingalls: No. They [settled there for life, but not before.] His father ran the C&O Railway and they built a spur up to this hotel in [the mountains of] Virginia and developed it and then my grandfather sort of fell into [running the hotel]. He was a New York lawyer but [had burned out from the stress]. He went down to the Homestead, that's this hotel, to recover and was—as he was feeling better and better he sort of enjoyed the—it's a beautiful place, all sorts of good country and good sports and he was a sportsman. And he was coming back from a horseback ride one day and was met by some people who said the person who was running the hotel had just died and he sort of fell into the role of managing the hotel and did that for the rest of his life.

Hsu: That's a really fascinating story.

Ingalls: Yeah.

Hsu: What sorts of hobbies did you have growing up?

Ingalls: Mostly I was a tinkerer so I mean I guess I played with trains as a kid but I also took things apart as much as I could and put them back together and tried to understand them and later on I built a lab in my basement. I remember I would walk to school and when I started thinking about mechanical things and stuff, electrical things there was—Thursday was trash day and I started noticing trash cans out as I'd walk on my way to school and then I'd see interesting things in them so I started coming back home with these things. And my parents explained that it's not okay to take things out of people's trash cans and I'd say, "But they're going to throw them away and they're so neat." So we negotiated an alternative solution,

which is I wrote up a little card that said, “I’m interested in electrical and mechanical things and if you have anything to throw away please call me up” and I went around the neighborhood handing these out. It was a bonanza. <laughs> I mean people had so much stuff and wanted to do this for a kid so that gave me hobbies plenty. And I was not particularly into sports although I really liked the outdoors and I wound up getting into horseback riding and stuff but–

Hsu: What subjects did you enjoy at school?

Ingalls: Well, you can guess sort of, math and science, yeah, and the school I went to had a nice shop and so I learned earlier than most kids to use torches and lathes and things like that and that turned out to be useful later on so I’ve always liked tools and working with tools. I like to build things.

Hsu: I think you mentioned in another interview that you were kind of like an inventor at a young age.

Ingalls: Oh, I did all sorts of—I mean I think of it as like any other kid; I did everything I possibly could but—so I built a bunch of things in the basement. So one thing I did which is maybe unusual is I played around with—I figured out how to make a carbon arc and I was always trying to melt things or blow things up, that kind of thing, and carbon arcs are cool, they’re really hot, but I also had a huge magnet that came from—I got it at surplus; it came out of a magnetron that was part of a radar system. And I noticed that in the magnetic field the carbon arc did all sorts of weird things and so I played around with that, and you can actually do things that are a little bit like plasma rockets in there. And then I started thinking of sort of what was making it do that and I wound up putting—I don’t know why one thinks of doing these things but I put a candle flame in the middle of the magnetic field and then put copper things on either side and put electrical signals into it and I found out that I could actually make it make sound; it worked like a speaker and it was essentially the essence of an electrostatic speaker that I did so weird things like that. And then I was playing—the other thing about the carbon arc was I noticed that—I thought that I could take it and focus it with a magnifying glass on the wall and maybe burn the wall, this—just as an experiment, not that you should do that but just as an experiment. And I got the glass at a different focus and I realized that I could see into the carbon arc everything that was going on in it so that was a lot like—as people do, projecting the sun and being able to see what happens on the surface of the sun. I actually wound up doing a science fair about plasma and things like that, but yeah, I just—I think neither of my parents really thought much about what I was doing in the basement and it was delightfully free so I just tried everything I could.

Hsu: They weren’t worried about you blowing up the house or burning it down.

Ingalls: No. I think I wouldn’t have done lots of the things I did if they knew because I got later on into exploding wires; this is where you get capacitors and charge them up and then you pull the wire across the contacts and it explodes. So in Virginia you could buy fireworks but in Massachusetts you couldn’t

and it was a no-no for me to work with fireworks in Massachusetts but this was another form of explosion which was just as interesting so—and I learned all sorts of things. I never thought of it as other than a wacky experiment and then I was in a bookstore up in Oregon and walked past a bookshelf and I noticed this title on the shelf, "Exploding Wires," and apparently there's a whole science about it, basically military projects I guess.

Hsu: How long did you live in Massachusetts or when did your family move to Massachusetts?

Ingalls: Let's see. As soon as my father got out of the [OSS], which would have been probably '45 or '46, we moved to Cambridge.

Hsu: Oh, and he got a job at Harvard or–

Ingalls: Yeah, and he had been at Harvard before [the war] but I didn't exist then but—so—yeah, so then I lived in that house basically through college. I graduated from Harvard and then I came out to California to go to grad school so—and I never lived again in Massachusetts but I loved it there. Cambridge is really nice.

Hsu: Since your father was already at Harvard, I guess it wasn't as difficult for you to get into Harvard for college.

Ingalls: Well, I don't know. I managed to get—I got a scholarship so I can't tell. I was doing well in school, right, and the interesting thing is that in four years there I ran into him on campus once because he spent all of his time around Widener Library and I spent my time over in the physics building and in the dorms.

Hsu: Right, yeah. I mean that is pretty fascinating that I mean he was a Sanskrit Scholar, ancient languages. Obviously, you had this affinity for science and experimenting and all these other things as a child. Where do you think that came from?

Ingalls: Well, he had—looking back I can see there was a lot more in common between us. He was a really smart man and he—I mean he was really good with words and language and he actually wrote his thesis about Indian logic so he understood about logic and—but his real interest was in sort of the flow of civilizations and how language changed through that and also all of the—there's quite a tradition around Sanskrit literature and the Indians were actually a lot more conscious of using their language carefully in poetry, they had theories of poetic allusion and all that stuff, but he also—he loved to do crossword puzzles, I love to do crossword puzzles, and he was fierce with them. I can remember when I was about, I think 14, we were alone in Virginia at the time and we got the Times, the Sunday Times and he said, "Oh,

a diagramless. Do you want to do this?" and I had never done a diagramless crossword puzzle; they're fiercely difficult and he just kept going through it. So that—the—sort of the logic puzzle together with the linguistic thing was something in common and we sort of came together around this. Later on we may get into this but I did some computer work to help him with doing some analysis he wanted to do with Sanskrit.

Hsu: Oh, wow.

Ingalls: It was a lot of fun.

Hsu: You started undergraduate studies at Harvard in 1962 and finished in '66 and you studied physics. What branch of physics were you interested in?

Ingalls: Pretty much general physics at the time. I didn't have a particular one. I took no courses in this but I followed as an enthusiasm the sort of attempts at controlled nuclear fusion and that continues to be interesting to me today. It's going to happen.

Hsu: Was your first experience with computers in college or before?

Ingalls: Yeah. It was during college, yes. So I didn't do anything in my undergraduate period until the very last year and the last year I had a course in FORTRAN and I thought that was cool; this is back in the '60s mind you. And before that in the fall I had taken a course with a great guy there who ran a lab that had an analog computer in it and analog computers are really neat. You get to wire up these functional components that will integrate or differentiate signals over time and you sort of connect them with like a pegboard—or a plugboard—sorry—and they're live; they're just doing what those circuits do and so it's a really visceral experience and they're connected up to—this was connected up to an oscilloscope so you could plot one signal against another and do things and I thought that was really interesting.

My one serious foray into hardware I actually built a—well, I read an article about how you could do analog-like things with digital computers and basically it was how you could make a pulse train that had a certain rate of pulses that represented a signal, and right at that time—I had played around with circuits quite a bit, probably for a couple of years before that. One of the things about living in Cambridge was that there was a great surplus electronics place behind MIT and I figured out how to get there and get things, and so I would get old circuits and play with them and learn about them. And right about that time Fairchild had come out with the first affordable chips, microcircuits, in little plastic packages so all of a sudden for a buck you could buy something that ran 20 times faster than the circuits I could get and it was reliable and easy to plug together so I got interested in building something with them. And in this course in electronics the professor who was running it had wanted people to do a project at the end so I got the

idea to do this thing a little bit like the Analytical Engine but a very simple version of it. So I built a thing that could compute logarithms and they're—and I can show you a picture of it; it's the most complicated thing I ever built and it was kind of a novel approach. And there actually was another company called Wang who—or Wong in the area and actually when I was done they—it turned out they were building something a little bit similar and so I would have worked for them that summer except I was going to Virginia but I wound up getting a—there was a scholarship and a prize that came with it for the—from the New England Regional Eectronic Manufacturers but that was a lot of fun. I've wanted since that time to do a simulation in Squeak or Lively of how that worked; that was a cool thing.

Hsu: This was the Wang company that made the word-processing mini-computers?

Ingalls: Yeah, and they also—they had a calculator that they built; it was like a scientific calculator so it could do sines and cosines and all those good things. Yeah.

Hsu: Then you applied to graduate school at Stanford in electrical engineering.

Ingalls: Yeah. The choice of Stanford—I actually—I applied to several and got into them—all of them and–

Hsu: Which other schools did you apply to?

Ingalls: I'm trying to think. Dartmouth was one and I can't remember what the other one was but I thought "Stanford's in California; I like beaches; I want to go there." I was done with Massachusetts. So I had no clue. I wasn't around the beaches in California. What I fell in love with in California was the Sierras but—yeah, so I came [to Stanford] in electrical engineering because that's what I was into then and got my master's there but also started taking courses in computers and that's—I pretty quickly sort of jumped into computers.

Hsu: Yeah. Around that time had computer science become a separate department or was it still a division in applied math?

Ingalls: It was a separate department at that point and I thought of transferring and I might have done that except that I got involved in starting a company instead.

Hsu: Right. Okay.

Ingalls: But I did—I took a course from Don Knuth and—yeah—and I had my—<laughs> it's funny—I had an un-favorite part of physics and my un-favorite part of computer science was numerical analysis and I had to take that course but I loved Don Knuth's work and—

Hsu: Which specific class did you take with him?

Ingalls: Just his basic—his first graduate course in computers, but what I loved about it was I always thought that computers were really cool and I liked how you could program them in assembly language and—but it seemed to me it was like a game, it was like a puzzle, and you could never get credit for that. And here was this genius who had taken it to the highest level and it was really intellectually respectable work and I got that connection from the kind of tinkering I loved to do with [serious] work so that really lit my fire in computers. And then he had a graduate seminar that was involved in measuring the performance of programs and I got really interested in that and wrote a program that would analyze other programs and that started this company that pulled me out of graduate school.

Hsu: That was just something that you happened onto and you had no plans initially to be an entrepreneur?

Ingalls: No, I hadn't, but I had—but I've always had a—kind of a practical streak; I want to do things and making money is a—it's a measure of practicality. It's the only commercial thing I ever did like that, [and a weather station I developed later]. I've thought of other things but—yeah.

Hsu: Was it difficult to decide to leave the Ph.D. program?

Ingalls: No. <laughs> I mean the things you do they're not difficult, they're what you want to do, so—and it sort of fell together. I can't remember how but somebody hooked me up with a lawyer and a business man to sort of get a company started and then I went around—so this program that I wrote would analyze the performance of FORTRAN programs and I've got some output from that that I can show you if you're interested–

Hsu: Sure, yeah.

Ingalls: It was kind of interesting because I went around the Bay Area trying to sell this. I had some successes but the big users of FORTRAN were all scientific users and their work was being paid for by the government. And my program was about how to make your programs more efficient and use less computer time and save you money. Okay, but the scientific users for the most part, they wanted to show that their computer was already too busy and so they were applying for a government grant to get a bigger computer. So [efficiency] didn't matter to them so much. But along the way I ran into a couple of

businesses that were actually trying to save money and their business software was mostly COBOL at that time. So I kind of rolled my eyes and thought well, I suppose I could do this for COBOL as well. <laughs> And so in the process I had to learn COBOL and not only that but learn COBOL in such a way as to be able to actually run with the programs, but I did that and it was pretty successful so. I can remember one sale—I went to—it was Safeway in San Leandro and they had—that may have been where they did all the processing for the Bay Area at that time—I don't know—but they had this one program that was using hours each week and I ran my program on it and it was spending 90 percent of its time in two lines of the program, which was clearing out this big data area. And so I said, "You know, you can save the cost of my program next week if you buy it" and it turned out that not only were they doing this thing but it was completely unneeded because they had a pointer to how much of the area had been used and they just had to reset that <laughs> so that was a big success. it was nice because in those days people—programs weren't live for people and my program sort of brought them to life by showing on the original source listing—there were measurement programs around but they would give you information in arcane core dumps and memory locations but this printed the results on the original source code. So you could just look at it and the person who wrote the program could understand what was going on so it was fun and I had a fun kind of—I think it was about a year or two when I would be the sort of traveling salesman and my business man would make contacts like cold calls and then I would go to places that were interested and it was fun to show to people because I invariably was able to tell them stuff about the work they were doing.

Hsu: The company was just you and a business person?

Ingalls: Yeah, and a lawyer.

Hsu: And a lawyer.

Ingalls: Yeah, and I was basically most of it. I mean the lawyer and the business man were really part time.

Hsu: Right.

Ingalls: I wound up selling that company to a company called Capex in Phoenix who—they were one of the companies that had these measurement programs that you couldn't understand so that was nice, and I had gotten a little bit tired of doing it and I got some money for the sale, which was nice. I didn't get rich but it seemed good for somebody who was just out of grad school, So the story goes on a bit from this. There was one fortunate thing that came out of this. this was in the days when there were various mainframes made by IBM and CDC and Univac and if you're selling a program like this you have to make sure it runs on all those. And so I would spend my night times sort of hanging out in these various service bureaus (you did it at night because the computer time was cheaper then) making my program work on

all these various [machines]. So I was at—one time at the CDC Bureau in Palo Alto getting my program to work and there was another guy there who was doing something with a FORTRAN deck, and I just started talking to him. And it turned out that his program was doing the kernel of speech recognition so it was trying to do clustering of phoneme sounds by their frequency distribution and he had the data in his program. And I said, "Oh, you want to find out what your program's doing? I'll run my program on it for you" so I did that and we talked a bit, and it turned out that he had been just hired by Xerox's new research center to do speech recognition so we chatted about that and parted but he had my phone number. So he called me back in a week and said, "Hey, how would you like to come to PARC and help me do this speech recognition?" so that was the segue into my time at PARC.

Hsu: Right. It was only after you had joined PARC that you sold your business.

Ingalls: Yeah, I think I was just getting rid of it at that time but—yeah, and that was okay with them but—yeah.

Hsu: You had already been thinking about maybe exiting the business at that time.

Ingalls: Yeah. I don't really remember. Yeah. What was going on at Xerox PARC pretty soon looked much more exciting and interesting and I was making money from the other programs—from the other work but this was a regular job and I liked it and close to home.

Hsu: How long were you on the speech-recognition project?

Ingalls: Not too long but long enough to do some fun stuff. Basically, when I arrived George—this is George White, was the person in charge of–

Hsu: What year was this?

Ingalls: –speech recognition. Well, it probably was '71, might have been '70. I'm not good about time so right around then. I guess we could track that down, but George had just taken possession of a Sigma 3 mini-computer which—so this—Xerox was not that much into computing at this point. I think they had bought XD—is—SDS and that's what had been given to George and it had a card reader and then George had a—one graphic display and out of that he was supposed to do a speech recognition system so it was not the kind of system that you'd like for doing that development. So what I did was using what I knew of FORTRAN I kind of built him a personal computing environment there where he could edit his programs, submit them and get them to run interactively and display the speech data on the screen and then do the—I had done a spectral analysis program with—for Fourier analysis before so I added that to it. So we wound up doing some speech recognition and this—and we had this sort of personal computing

environment hacked together from stuff that—which nowadays you just have all on your laptop but this was spread around this hardware.

Hsu: And the idea to do this kind of personal-computing environment was that because you had already been talking with Alan Kay or was that something that you had come up with independently?

Ingalls: No. It just seemed like the thing to do. I've always wanted to get my hands on stuff and it seemed to make sense to do this and the fastest way to work on speech is to get it so that you can edit the programs there live and not have to punch cards and do that. Yeah, so I think that was just a natural thing that I wanted to do and I think George wanted something like that so it was pretty obvious what to do.

Hsu: What were the components of it? There was an interactive text editor?

Ingalls: Yes. I wrote an interactive text editor and I modeled it after the one—there was one at Stanford—so I mean I didn't have any of that code but that gave me an example of editing commands that are convenient so there was that, and then I did the stuff that was needed to go through the operating system so that the minute you were done editing something it would run it and then get you back to edit it when you were done, that kind of thing. I mean most of it at that time that I was playing around with was FORTRAN because that's what I had in this thing but I had a lot of fun with—there's a system called META that was done by by Val Schorre I think in 1962 or something like that but it's a compiler-compiler. And I had done a version of that in FORTRAN so that you could do other languages in FORTRAN but for the speech-recognition work we just kept it to be in pure FORTRAN.

Hsu: What other languages were you familiar with?

Ingalls: Well, META, if you can call that a language; it's not—it's a programming language but really a compiler-compiler. I had played with BASIC and I had a chance to play with APL some. A friend of mine, Ted Kaehler, had worked at IBM's lab down on—in Palo Alto and I got a chance to go over there and play with that, and then somewhere I had a chance to use a timesharing terminal and I used APL on that and I guess also a chance to use PL/1 so—and then I had done some assembly language coding. There was a little HP mini-computer at the computer-science department at Stanford that I had programmed by assembly language and then I had done a—this Fourier analysis—I did a Cooley-Tukey program in assembly code for the F360 I guess it was.

Hsu: You mentioned Ted Kaehler. Had you known him before you came to Xerox?

Ingalls: I'm fuzzy about how we came together. I don't really—I don't remember exactly. I should get in touch with Ted and see what he—Ted remembers everything; he's much better than I am about that. And I don't actually remember how we came together, whether Ted independently came to Xerox and we met there or whether he was also a student of Don Knuth's. I'm just not sure. Yeah.

Hsu: Okay. Let's talk about your first meeting with Alan Kay. What was that like?

Ingalls: Well, as a part of working on the speech-recognition work with George White I had an office next to George's and it turned out to be across the hall from Alan Kay's office. I realized after just a few days that I was often more interested in the conversations that I could hear Alan having than I was in the work I was doing. All the things he was talking about sounded interesting and as you know from Alan he's—he has such a depth of interesting sidelines to every intellectual topic so eventually I just went across the hall and said, <laughs> "What are you doing? I'd like to know more about it." And I think that was the point at which I had done most of what I needed to do to support George and George could go on doing the speech part that I didn't have a real passion to pursue. [Alan and I] hit it off together pretty well I think and I know that—well, Ted and a couple of other people were around at that time and I know that one of the things that Alan was talking about was doing a small personal computing system, and that really excited me because I had tried to do that with a roomful of hardware and the idea of doing something that was small and personal and maybe even portable was really exciting. And Alan had also the idea of how to do a small language and I had thought about that from my work with META but Alan had sort of a more concrete picture of it. So we started this conversation about what could you do, how could you build a small system and what would be a good language so there were a bunch of conversations around that all of which were fascinating and eventually got us to—into Smalltalk.

Hsu: You mentioned that Ted and a few others were already there. How big was the group at that point?

Ingalls: It was not big. I'm trying to think of who actually was there. Adele was not there yet. I don't think John Shoch was there. I think at that time it was maybe just me and Ted, but when I looked at Alan's history he mentioned somebody else so there may have been somebody else but I remember—well, we talked about languages that would be neat. We wanted to be able to build a language that could easily model different things in the world so the notion of classes of objects and if you were going to do something that was a graphical system you'd want to be able to model things like points and rectangles, and there had been some extensible language work done but most of them got—sort of got more and more complicated because of being procedure oriented. If you try to do an extensible language that way it does—there's a an N-squared complexity problem you run into with the more different kinds of objects you have. And Alan had this idea of doing things by sending messages and I don't know exactly where he got that from but I know he—that's what he wanted to do, and it was clear that if you could build a system that way that—actually where that was the fundamental way of programming—that you could have many different kinds of [objects] without having this N-squared problem. So all of the code that went with rectangles would be in this class and all the code to do with points would be in this class, all the code to

do with drawings would be in that class, and so how did it work? I can remember Alan said, “You ought to be able to do another beautiful kernel like McCarthy’s Lisp kernel for a language like that and he worked on it really hard for about a week. I think there were a number of days he didn’t come in to work <laughs> and then he showed up with a sketch of how to do an interpreter for a language which became Smalltalk-72.

Hsu: Right. This is sort of the famous bet that he made with you that he could design a language in one page?

Ingalls: Yeah, pretty much, yeah, and it was a really unusual language and I—it’s the coolest scripting language I think in the world and if I were teaching a computer-science course I would have all my students implement a language like that because the way that it worked—well, it worked a lot like META actually, so it sort of was parsing while it—the program was parsing while it ran. So conventionally when you call a procedure you give it arguments and then you—the procedure computes a result and gives back the result, and in Smalltalk-72 when you called a procedure the procedure got not just an argument but the entire input stream, whatever else there was in the program, and it could then decide what to do based on that. And it made the kernel interpreter incredibly simple and it subsumed all the—there was no compiling to do; it just ran so in that sense it was fairly Lisp-like but it had more generality to it.

Hsu: So that was Smalltalk-72. That very first sketch that Alan sort of wrote up and both he and you had mentioned that you then took that sketch and you wrote the very first version of Smalltalk in BASIC?

Ingalls: Yeah. I mean there it was and you could sort of look at it and imagine how it might work and so what I immediately wanted to do was to make it work, and at this point I was wanting to work at home because I had a young—a very young child at home and the only thing that I could get to work on at home—to work with at home was a timesharing terminal that ran BASIC, at least the only easy thing, so that’s what I did. So Alan had gone away for about a week figuring out how to do this thing so I went away for about a week and wrote a BASIC program that essentially did the kernel interpreter and made it actually run, and I remember—I can remember it running six factorial and it all worked so that was our proof that you could—that this kind of interpreter could work. We hadn’t tried out anything with points and rectangles and graphics or any of that but it was clear that it could work so that said to me I could take the next step and try to start to do a—[serious] version in assembly code because that’s what we would have. And there was—I think at this point—and Alan was negotiating to get Altos built and we could sort of see this future that—I think at that point it was known that the Altos would be emulating the Data General Nova instruction set and there were a bunch of Novas around Xerox PARC. So I got a setup where I had one of the Novas reserved for me with a telephone hookup that I could log into from home so then I could now do that work at home. I started doing a production version of this and there was lots more work to do. I had to do symbol tables and garbage collection and all of that stuff but we got to where that ran, and the timing was nice there because I got that running right at about the time when the Altos were ready to start trying stuff out and then we had graphics to play with and—yeah. And by that time Ted Kaehler was

involved in the project and Diana Merry so Alan had Diana Merry working on some of the graphics and [text, and] Ted did the turtle line drawing and worked with me on storage management.

Hsu: Wow. I'm trying to remember the–

Ingalls: Ted was—somehow he had been a student of Don Knuth so I know that's how we came together and that's how we both were working on storage management because that's—a lot of Don's basic data structure stuff went into Smalltalk-72. It was great. This project was—it was—it just generated perfect problems one after another, archetypal Don Knuth problems.

Hsu: So it was just a very natural application of all those principles from his course?

Ingalls: Yeah.

Hsu: So Diana had joined the project at that point. Do you remember sort of how Alan had recruited her?

Ingalls: Not at all, but it was a great group, a bunch of wonderful personalities. Everybody was so supportive of everyone else.

Hsu: And so you, Ted, Diana were the only sort of permanent members of the group 'cause I think there were other people that were graduate students that were working with them.

Ingalls: [To be clear, Ted and Diana were the people who worked with me on the St72 interpreter.] Ted was a graduate student in there but somewhere in there he got his degree from Carnegie Mellon and then was permanent and I don't remember exactly when that happened.

Hsu: How was it sort of decided who would work on what? Was it just whatever people were interested in or–

Ingalls: I think we just sort of fell into a natural organization. So Alan knew what he wanted, I knew what I wanted for the architecture and the engine, and then we worked together; so Alan worked with Diana, telling her what he wanted the text to do and getting scrollings and working—things like that but she would come to me with problems and I'd help and Ted would—we'd all help out. And Ted was just naturally good with all sorts of detailed work which—if you want to do line drawing on a bitmap display and nobody's done that before, it's detailed work and—I don't mean nobody's ever done that before but it was new territory, but I think I coordinated that pretty much and got things to fit together.

Hsu: Yeah. I think in retrospect Alan sort of calls that original version of Smalltalk that I guess ran either—it was either the BASIC version or the Nova version he calls that Smalltalk-71 and then the one that ran on the Alto was Smalltalk-72? Is that categorization kind of correct or is that kind of a fuzzy–

Ingalls: Well, no. No. I felt that the original BASIC version and everything that came after as basically being Smalltalk-72 from the get-go, and Alan had—Smalltalk-71 never got done, never ran, but it had been a design that Alan had that was different in a couple of ways but the same basic thing.

Hsu: Oh, okay, yeah, 'cause I think I remember [in] his history of Smalltalk paper he said he was working on an iconic language before you guys made the bet, so that would have been the design of '71?

Ingalls: That's my guess, yeah. So I never really looked at it seriously or understood it . Alan would know about that and I don't really. Yeah. I think he had some ideas there that never got done, but Smalltalk-72 wound up supporting the kinds of things he wanted to do and it was iconic to a degree. So we invented various special symbols to look more like what they meant.

Hsu: Oh, okay, yeah, special characters that were in the language. Yeah. Earlier you mentioned how the language essentially parsed whatever input was coming in. I think Adele Goldberg has told us that that made the language very context sensitive. Did that make the language harder to use for certain users?

Ingalls: No, I don't think so. It made it a better language and a worse language and I'll explain that. Okay. So when a function got called it got control over the entire rest of the program and so it could parse it as it wanted. And this was—we could not have done what we did without that ability in the following way, which is, it meant that you could—just with this function-oriented language we very quickly saw how to make a real message-oriented language. There were a couple of primitives for operating on the message stream that would match a token and so it made it possible, and I can show you some of the programs at another time. Well, to take the example of Seymour Papert's turtles, if you were writing the code for the turtle you could do—you could test the message stream and see if there was a "Go" or if the next token was "Turn" and then if it was "Go" then you would do the following thing, which is draw a line. If it was "Turn," then you would do the following thing, which is to change the direction and if it was "Print" do the following thing, which would print out the location or something. So without there being any message apparatus in the underlying engine this allowed you to write programs that were essentially message-oriented so we right away fell into that pattern and I think that was basically—Alan had been exploring that kind of pattern with Smalltalk-71 I think. So that gave us a lot of experience writing programs that were message-oriented, that were object-oriented. It also made it easy to make idioms, in other words, because you could put other words in there that were needed, like for instance on a for-loop you might just give it a couple of numbers for the start and the end but you could also put in what we called ["noise"] words, which are ignored tokens, so you could say, "for i from 6 to 12." You don't need those extra names but you put them in and it reads so that you can understand it. So that was—that all fed into the later Smalltalks in terms of the style and what made them more readable. That was the good

side. There was a bad side, there were—well, a couple of bad sides. One was it ran very slowly because—the program wasn't just executing, it was having to parse on the fly—so it was not compiled, and in fact it was not compilable because you couldn't tell what a pattern in the program was going to do until it actually got called and got parsed. And that also meant that you couldn't read it! Although we had a style of writing that the programs were all generally readable, you could come to a place and you would not know what a given token would do. We had one case where it was a real problem, which was we were trying to prepare some demo and we brought together programs from three different sources and one of them changed the definition of how numbers parsed and then the other ones wouldn't run at all. So it's known that this is not a good design but it was a wonderful [artifact] for research; it just—it made the language like putty and we got to explore how we would like it to be as users and that then informed the later Smalltalks.

Hsu: Right. I think Alan wrote in his paper that it was too extensible; it was great because it was so extensible and you could do whatever you wanted and it was almost like every different function had its own syntax that it would define but they were all different and so they could be totally inconsistent, which made it completely unreadable.

Ingalls: Yes, it could make it completely unreadable but the truth is with a bit of care it was actually very readable so it was a very natural language. Kids [seldom] had a problem with it but they weren't putting together big pieces of software.

Hsu: Right. Oh. So you mentioned the kids. How much were you involved in those experiments with the kids?

Ingalls: Not a lot. So I did—it was great having the kids around and I would see what they were doing and sometimes I would answer a question or maybe take somebody and get them started, and Adele really ran that show and others of us would sort of feed into the—as part of the instruction, but she had the real background. I was interested in seeing what the kids were doing and then in helping them in a small way but I wasn't in on any proper curriculum for the classes or anything like that. When something that was needed, I would right away figure out a way to do it and support it.

Hsu: So they would generate things like feature requests for the language and you would implement them for them.

Ingalls: The usual feature requests and bug fixes.

Hsu: So of course by that point Adele had joined the group. What was it like when she joined and how did that sort of change the dynamic of the group?

Ingalls: It didn't change the dynamic that much. Alan has sometimes said that he brought the group together by finding people for each part of his passion and Adele met the educational-engagement part. I never felt it was anything new to us because I felt that we were doing this so the kids could use the computers and could do cool stuff. And then there was Adele introducing kids to it and doing cool stuff so it didn't change the dynamic much. We had a fortunate set of personalities. There was very little ego and I think Adele had her purpose of how things ought to work for doing an interesting educational experiment. I was very clear about how the engine had to work and how Ted's work and Diana's should fit in, and Alan had his overall theory and passion and he would give public talks that pulled this stuff together and made demands on us. Alan's demos were a great part of how our group worked. We often talked about it being show business and it really was. There's a performance coming up and we've got to get this and that ready and there was always—Alan had a—kind of a flame out in front of what [he] wanted to happen and so it caused us all naturally to get behind that.

Hsu: So was that sort of the way that he drove things was "Oh, we want to do a demo and this demo needs to have X, Y, Z" and that was sort of the vision that everybody was aiming for?

Ingalls: Yeah, I think so. I mean it's—that's my general feeling about it, yes, although it kind of worked both ways. The other thing that would happen is that we would just do cool stuff and he would see it and pick it up and it would become a part of his demos.

Hsu: What was he sort of like as a leader or I mean I guess "manager" is maybe not the right word for what he did but–

Ingalls: [Alan was the visionary. He set the course and he was responsible for support. So we needed to support him in that role.] He was pretty much a hands-off manager but he was interested in everything and he understood what each person was doing and he would challenge you but—and also feed you good ideas. He read really widely and knew all the work being done in various systems and other languages and various solutions to do this or that in other systems. We would pick up bits and pieces that way.

Hsu: How did you as well as the other people interpret Alan's vision and what was it about that vision that sort of you were so passionate about?

Ingalls: I think the—just the feel of real personal computing. I mean it wasn't—it didn't exist before really. You had timesharing systems where you could have your own individual thing going but the vision here was to have real graphics and dynamic graphics and a language that you could see and change. And somehow from the very beginning we all thought in terms of everything being live and there were other systems that were like that. The main other system like that in the world was Lisp and those systems were live but there wasn't much being done in Lisp with graphics at the time, and Alan totally had that

vision and there was—this was also just when bitmap screens were coming into the world. That really happened right around then and that suddenly made all sorts of things possible that weren't before so there was just a host of things waiting to be invented, how to do this or how to do cool stuff with text and bitmap displays and—

Hsu: Yeah. So then obviously graphics was always a central part of that vision.

Ingalls: Yeah. Well, it's so much fun basically and especially it was all just happening then, it was all new, and graphics and sound as well. We did music synthesis on the—we even got that working on the Novas before the Altos were working and when the Altos got working they were microcoded and so that gave us quite a bit more power so we could do on the Altos ten or twelve voices in real time synthesized and we—so then in Smalltalk we could write a music editor and do that.

Hsu: So then the transition to getting Smalltalk-72 working on the Alto was to rewrite the kernel of the language in the Alto's microcode?

Ingalls: [Not at first.] The first Smalltalk-72 that ran on the Alto, most of the Smalltalk language kernel just continued to run as Nova code; we just moved the Nova code program over and actually the graphics was done in Nova code and it wasn't super fast but it was fast enough to try out various things. We had a text editor working and we got Turtle [graphics] working.

Hsu: Because you had known that the Alto would emulate the Nova and you had the Novas, you had planned it out that way that you would write it for the Nova first and then it would all port naturally over and then you could rewrite bits and pieces of it in microcode as you saw performance improvements.

Ingalls: Yeah, and the early Altos didn't have a lot of extra space for microcode anyway so—

Hsu: So then which bits did you end up rewriting in microcode that were important?

Ingalls: Well, so coming on—into the next generation there was the move to Smalltalk-76 and then partway in there we also had Smalltalk-74 and we can talk about these various parts, and then I did BitBlt <pronounced Bit Blit> kind of independently in there. It first ran connected to Smalltalk-74 and then in Smalltalk-76.

Hsu: Right. Okay. That's right. So we'll get into that a little bit later. You mentioned earlier that the language being live the way that Lisp was live, I guess another word for that is "dynamic." So was that the

idea that the user could interactively change anything about their environment? Why was that such an important part of the vision for the language and the environment?

Ingalls: Well, let's see. There are a couple of things about it. One is just in terms of the work-and-reward cycle of doing programming; if some—if you make a change and you get to see its effect right away, then if you've got to make five mistakes before something works you're done in a minute instead of you're done in two hours. And so this is a really powerful thing in live computing about a message-oriented system and about object-oriented programming, which is that essentially every time any procedure is being called in the system it's being looked up whereas in a conventional compiled system all of that stuff is done ahead of time by a compiler and a loader and you cannot change it. So when you make a change in your program, you have to go back and recompile everything and reload it and stuff. But if the system is put together by sending messages you can change it on—right in place and the next time it executes it'll do the new thing without any problem so this is something that just couldn't be done other than this way and if you organized things by sending messages it was effortless.

And so an interesting thing happened with—our group was the Learning Research Group at Xerox PARC and there were a couple of other groups and there was the Computer Science group [Lab] and they were doing serious programming language research and work. They worked on BCPL first and then Mesa and these were compiled languages that you had to compile and load so—and we had these—there were these meetings at—that were held by that lab [CSL] that was under Bob Taylor called Dealer and it was called that because each week somebody got to be [the] dealer and sort of talk about something, but they were—it was very much of a—sort of a brown bag, bring-your-lunch and we'll talk. And so people would talk about good ideas and our group, we were at heart all computer scientists; it's just that we were sort of the renegade kids' group <laughs> and—but we'd come up with ideas and—or they would come up with ideas and we'd go back and that afternoon get it working because [although] our system ran incredibly slowly, Smalltalk-72 at the time, but the turnaround time—if you made a change, the turnaround time was about five seconds before you had the result whereas they were working with a system that had a very long turnaround time. So frequently we could try out some cool hack and have it going that afternoon where they just wouldn't even bother trying because it would take too long to try so—and that was a nice thing. I think we sort of pushed them in the creativity way and they challenged us in a kind of serious software way, a nice relationship.

Hsu: Yeah. Can you talk maybe a little bit more about that kind of collaboration that the LRG had with the Computer Science Lab?

Ingalls: Yeah. Well, so that part was social and there were various people that kind of—there were various bits of respect that went across that boundary and the—at—but not at the level of collaboration so much as in the—it was more like communication and social things, but there was serious collaboration in terms of the underlying systems there. So the Computer Science Lab did do really serious work about the—getting the internet to work and getting—excuse me—the–

Hsu: Ethernet.

Ingalls: –the Ethernet and getting the operating system that was first working on the Novas and then the operating system that ran on the Altos—they ported that over also—and the printing facilities which—all of which we used and that—so that was great. I don't think they used much of our stuff except—well, BitBlt is a big counterexample to that, but I think we helped to generate ideas in the user interface so we showed a lot of nice user-interface stuff [windows and menus, for instance] that we explored in Smalltalk before they got that working.

Hsu: So you guys had already prototyped it in Smalltalk and they took those ideas and implemented it in Mesa or Cedar or whatever else–

Ingalls: Right. So very early on we got a windowed user interface working. That was something that Alan worked on a lot together with Diana Merry before there was even BitBlt support for it.

Hsu: Alan getting sort of that iconic—well, that's not the right word—the classic overlapping windows, menus, interface, how much of your work was involved in getting that graphical interface up and running early on?

Ingalls: So I worked on all of that. The [aspect of the interface that] was distinctly mine, was the work on BitBlt and that that enabled later on menus but the—we had—it's interesting how the menus evolved. So we did—in Smalltalk-72 we just had a simple sort of a—what people call now are REPL windows, a simple window where you could type expressions in the language and make things happen and do your programming there and then–

Hsu: Oh, you mean REPL, read-eval-print-loop.

Ingalls: Yeah, and then John Shoch, who had come onto the group, did a simple sort of code editor and I worked—several of us worked with him on that but he had made one where there was a list of commands on the right, which was essentially a static menu but it was a part of the box, so over there you could select in the code and then you could [click on] "Replace" and it would do that or "Insert" and that kind of thing. And when I got BitBlt working, which we may talk about, it was clear that we could do that kind of a menu of commands and bring them—pop them up and have them disappear anywhere we wanted to use them [without requiring any real estate.] So there was the basic idea of a list of commands which was pretty simple that you can point out with a mouse and then the idea that you—the ability to sort of pop that up without destroying what's underneath became easy to do with BitBlt. I mean it could have been done anytime but it would have been really hard without that kind of help.

Hsu: Right. I see. So we mentioned the word “object oriented” earlier. What does that term mean for you or what did that term mean for you then?

Ingalls: So “object oriented” to me means that processing happens by sending messages to objects . It allows the system to be partitioned according to different, what used to be called data types but we call them classes now, and if the code is separated that way it makes the system much better organized and you don’t run into this problem.

An example problem with not having things organized that way is, supposing you want to add some new thing like rectangles to your system, a data type of rectangles. Well, you want to be able to print them and so to do that you—that would mean you have to go to the system print routine and edit it in to give the special code for how to print a rectangle, but the—there are several problems with this. The first is that all of a sudden you’re having to edit some big system procedure that knows how to print all the other things in the system, so it’s complicated. Another thing is that you may make a mistake while doing that in which case you crash the whole system because that routine no longer works and everybody else is depending on it. But if you organize things by sending messages then all the operating system—all the kernel has to do is to send the message “print” to whatever object it is so that part of the system is really simple and nobody needs to mess with it. And then if you—in describing rectangles you just write how to print the rectangle and you’re responsible for that code. If you mess up doing it your rectangles won’t print but everything else will still work just fine. So that problem of having the central routines have to deal with all the data types is a—it’s a scaling problem, which is you’ve got—and you may have several of these things that have to know about everything else and so it’s really an N-squared complexity problem and if things are message-oriented it’s not—scaling just goes as the size of the system. So that’s [what “object-oriented”] means to me and there’s—there are a bunch of sort of deeper aspects to it but the feeling we got as soon as we got Smalltalk working—it’s sort of not exactly an emotional feeling but [a sort of tactile] feeling—it’s like having a bunch of trained animals. You have these various parts of the system and you give them each their proper behavior and they just can then interact and it’s just a very natural way of thinking about things, and that whole feeling goes together with this live [situation] where things—everything on the screen—is live and you can change it while it’s running. And that grew into the whole field of object-oriented programming where you see it now. I mean there’s C# at Microsoft and Objective-C at Apple and those all grew. There’s a direct lineage of Objective-C from Smalltalk.

Hsu: Yeah. I think one interesting difference about sort of yours and Alan’s view of objected-oriented programming versus I think what a lot of people often think of it as, is [that] message passing is for you the central, core organizing principle. Other people sometimes think of inheritance as the organizing principle but Smalltalk-72 didn’t even have inheritance. But that was on your list of things to do. When you were working on Smalltalk-72, why was that not a big priority for the language at the time?

Ingalls: It’s a good question. I–

Hsu: Simula had it, correct?

Ingalls: It did, yeah. I think there are a couple of reasons but I think the major reason is we didn't need it at first and if there's a big comparison to be made between what we did and what Simula did is, we got stuff running and we got it running dynamically and things could evolve really quickly and be done really simply so that was kind of our focus. The Simula work, a lot of it was really good, deep stuff but we got a huge amount done in a very short time by sort of taking a simpler path so we didn't need it for what we were doing. We were doing quite simple stuff, which is graphics, kids' programs and the support of the system itself, which was pretty well organized and therefore pretty simple. And another thing is that certainly for the kids we worked with, inheritance is another thing you got to learn about, it's something that goes on that you can't see, and it's—and even if we had had it I would have tried to keep it out of the kids' area frankly. I don't know how everybody else feels but my guess is they'd agree.

[Inheritance is] useful and we exploited it completely in Smalltalk-76 and -80 and there's a wonderful play between—we can talk about this—there's a huge synergy between inheritance and the polymorphism that message passing gives you. But it's also the case [that] if you're just teaching somebody about inheritance, well, then pretty soon you wonder well, gee there—there's—you also want to get into multiple inheritance and it just—it can get very complicated and there's a much simpler world that I think is appropriate for starting out for teaching kids.

By the time Smalltalk-72 was getting mature, we had the whole subscripting package for arrays. It had a lot of nice features, so you could do subranges and replace subranges and all this stuff and we wanted to do the exact same thing for character strings. So there was a piece of code that was in the character string place that we just copied directly into the arrays to keep them consistent and they were the same and so it behaved as though they both inherited from something that knew about subscripting. So it was clear that we wanted that, we as computer scientists, but the kids [didn't need to] think about that.

Hsu: You mentioned also that you actually were glad that you did not know Lisp when you implemented Smalltalk for the first time and you felt that Smalltalk actually was better for it.

Ingalls: Well, who knows, and I don't know what I was thinking when I said I was glad but things went the way they went. I can imagine some things—if I had been into Lisp—I'm just not sure. We learned so much just by doing what we did with Smalltalk-72 and it might not have even mattered if it was based on Lisp, but the thing that—the thing we did learn is this style of how to code with dispatching on messages and just describing objects that way and the sort of [noise] words that made things more readable.

There came this time when I was trying to figure out how to make things run faster and it was—well, this is rambling a bit but we did this system called Smalltalk-74 in between and we can talk a bit about that but it was sort of like a somewhat better job of Smalltalk-72, but through that time I was kind of distracted

because it wasn't what I wanted; I wanted something that would be an efficient compilable engine that would still have the nice properties of Smalltalk-72. And I—it just kept not quite coming together and then all of a sudden I saw how it could all come together with a keyword-based syntax which preserved the nice quality of Smalltalk-72 and its [noise] words so that you had a much more readable separation of parameters. And I have a feeling that to have done that with a Lisp background I think I wouldn't have ever maybe done that but who knows. Now there's another way in which—which isn't exactly from Lisp but there's a germ in there, which is at that time I knew that Peter Deutsch had done a byte-coded interpreter for Lisp and I didn't really know anything about it except this phrase "byte-coded interpreter" was bouncing around in my head because this was by the time that we had access to the Alto microcode. And so making a byte-code engine made a lot of sense because you could have—you could get the compactness of compact coding and yet have the microcode speed to interpret it so I had that going over in my head and that's—that sort of led me to the design of the byte-coded virtual machine that became Smalltalk-76.

Hsu: Oh, okay. I see. Yeah.

Ingalls: We can talk about Smalltalk-74 and there are some advances that were made there but it—about the only thing we ever really did with it was to simulate Smalltalk-76 to get it running.

Hsu: So the ideas for 76 were already brewing while you were implementing 74 essentially?

Ingalls: Yes. So Dave Robson did the Smalltalk-74 work pretty much so I didn't do much of that.

Hsu: Okay, yeah. So then what were the—so, I guess let's talk about Smalltalk-74 then. What were the major advances in 74 as sort of an interim between 72 and 76?

Ingalls: So the main thing as I see it was that we had a proper sort of message object. You know, in Smalltalk-72, when a function started running, it came along with a message stream, which was just basically a pointer to some code and an index. And we had primitives for accessing that stream. So there was a primitive that would give you the next token. There was a primitive that would match the next token against something, and things like that. And in Smalltalk-74, those didn't need to be primitives, because you could actu—there was this other object which represented the message stream. And those functions such as match the next token or pick up the next token or evaluate the next token, those could all be written as more Smalltalk. This made it more general and more [explainable, but it did not make it faster.] And we made classes real objects. Classes were real objects before, but in Smalltalk-72, you couldn't send messages to them. And I, honestly, I don't remember whether we had that working in Smalltalk-74 or not.

Hsu: Yeah. So you mentioned that Dave Robson did most of that implementation work. But the ideas for those designs, did they come from you, or from other members of the group?

Ingalls: We talked together about it. I mean, I think the main people talking about that were Dave, me and Alan.

Hsu: Okay, yeah. And so we've been skirting around BitBlt also. BitBlt also came around this time, so let's dive into that next. So tell us about how that first started coming about. The idea for that.

Ingalls: We had a bunch of separate pieces of graphics code in Smalltalk. So we had Diana's work on text. And her work on text, there was an existing text primitive in the Alto, so she didn't do that, but she did do the management of text frames, and having multiple text frames and being able to scroll, and being able to move things around. And as a part of doing that on a bitmap screen, she had to deal with all of the problems of, if you want to move things over three pixels, underneath it all, the graphics is oriented on the Alto [in]16-bit words. And so there's a huge amount of shifting and masking that you have to do to make that work. So a lot of work. So she had a big pile of code that dealt with that. And then Ted Kaehler had written the code for the Turtle line drawing, for line drawing. And it's the same kind of problem of having to deal with arbitrary alignment in 16-bit words. Except more oriented towards line drawing so that where you're just doing a couple of pixels here and then moving to another place. And so I started to think of how could we pull these things together [for a new kernel (St-76)].

Hsu: Oh.

Ingalls: I came up with the idea, and I've got a picture of it that would kind of illustrate it, but there were, first of all to do all the shifting and masking in one place, and to come up with a routine that's general enough to do large areas or small areas and to do the shifting and masking needed. And put that all together in a way—so it needed to be parameterized so that it could do each of these functions. And it also wanted to be able to do any of the store modes, whether you wanted to store black into the destination, or "OR" it, or "XOR" it, so those were all things that we used in different cases. [I wanted to craft] a function that was small enough to go into microcode that would do all of these things. And I also came up with a way to do it without needing any temporary storage space. So some of the routines that had been used in Diana's text frames required fairly large temporary storage. And I figured out this way to do it without any, except for one register. [That made microcode a possibility.] So that we then tried out in Smalltalk-74, for the first time. [Actually I] first simulated it in Smalltalk, I did. And then I learned enough Alto microcode to do it, and wrote the first Alto microcode for it. And it was [amazing], because in addition to things like the scrolling of blocks of text, and line drawing, we actually could do individual character placement. And that allowed us to have more general text fonts, and have fonts that were stored compactly because you could pick up a little piece from anywhere. So that was BitBlt.

I remember we gave our first demo of it to the guys in the Computer Science Lab, and the particular demo that struck them was, I had—at that time I did a pop-up menu. And because it was now easy to pick up a piece of the screen and save it in a Smalltalk object, and then plop a menu down, and then put them back, it made it possible to do this thing which seemed like magic, which is boom! there's something on the screen and away it goes. And I still remember giving the demo of that the first time I did put up a pop-up menu. Peter Deutsch said, "Did you just do what I think you did?" <laughter> And that was a case of really great tech transfer between the two Labs, well, and all of Xerox actually. Because I remember the next day Smokey Wallace came over from the Product Division, and he said, "Hey, I want to know about BitBlt." And it made the difference that I had, you know, I mean, I was working in the Smalltalk group, and we were not considered to be product-y people, but this was [serious] technology. And I had taken it, not just to running in the Nova code, which might have been okay, but it was actually microcode and it ran really fast.

One of the cool things about microcode is that you have a chance to execute several instructions between memory cycles of the host machine. And it was perfect for BitBlt, because the kinds of shifting and masking that you had to do could [often] be done in the time between memory cycles. So a lot of the graphics operations we wanted to do were complicated, because they had to be moved over like seven pixels or something. But that could all be done under the shadow of the memory cycle, so that you were able to move things at full memory bandwidth, even though they're not placed right. The microcode architecture says to you, if you can think of a general enough small kernel, then you can put that in microcode and it'll run five times faster than anything else. So it's really motivating. And that also drove my wanting to do the byte-coded virtual machine for Smalltalk, because I could see that there's another case where we're interpreting instructions, and so the time to pick up the next byte and interpret it and figure out what to do can [be mostly] hidden under the memory cycles. So it's good for graphics and it's good for language interpretation. It lets you tailor the machine. It's like giving a new instruction set to the machine.

Hsu: Yeah, I mean, I think you've called that a graphics kernel. So to you a kernel is anything that can encapsulate some fundamental—well, how do you define a kernel? How do you—what is a kernel to you?

Ingalls: Well, I guess a kernel is kind of—it's a relative term, so I think of it as being a bunch of functionality that goes together and it produces a new—not quite a new metaphor, but a new way of operating. So in the case of BitBlt that was a kernel that meant that you could suddenly work with bitmap displays in a very high-level way. You know, you say, "Move this rectangle of stuff from here to there," and you don't have to worry about what the pixel alignment is on the word, so it sort of civilized that piece of the hardware. And the kernel of the Smalltalk virtual machine basically said, "Here's a piece of code that will make all of this high level code work as fast as it would have if the machine had been designed with that instruction set. So it sort of this allows you to work with sending messages to objects where you couldn't do that before. And you can imagine other kernels, such as for storage management, garbage collecting, that kind of thing.

Hsu: Mm hm, hm. And I mean, BitBlt ultimately is such a fundamental component of 2D graphics. I mean, eventually it was described in the graphics textbook that—was it Bob Sproull?

Ingalls: Bob Sproull, yeah.

Hsu: And the other co-author, whose name I can't remember, wrote.

Ingalls: Was it Newman?

Hsu: I think so, yeah. Yeah, yeah. I mean, how did you feel about your work being sort of such a fundamental aspect of graphics that would essentially be learned by all future people doing graphics?

Ingalls: Well, it felt great to have participated there. I mean, I had something real to contribute. And it was—it had a really natural evolution, which is we had these three different things, character generation, line drawing and moving big areas. And you know, if it wasn't me, somebody else would have done that. And there was similar work going on. You know, I know the Atari had functions like that which they were using for various animations. And that's another thing this enabled us to do was animations. We had done specific animation microcode before. Steve Purcell did a pretty neat little animation package that we had as a part of Smalltalk. But as soon as BitBlt was working you could do all of those things at the same speed. But with a lot more generality. Yes I felt good about it.

Hsu: So this, I guess this next question is maybe for our more general audience, but how would you describe how BitBlt works to a Museum goer? A visitor to our Museum?

Ingalls: Well, to a Museum goer, nowadays everybody assumes things. You know, we live in a world where you can do anything graphically on today's machines. And it's simply the ability to move pieces of an image from one place to another. And it lets you do things like painting, where you're stamping a paintbrush along here. [It] lets you do things like text, where you stamping letters along there. It lets you do things like scrolling, where you're moving this and that. And nobody really thinks about, nor should need to think about, nowadays, the problems of getting that to map onto the memory representations underneath. But it's basically, it's an image manipulation primitive. And there are other image manipulation primitives that you can imagine. You can imagine something that's completely driven by curved outlines, which would be more general in a way. And if you could make that run fast, that could be a really nice graphics design. But BitBlt takes care of the specific case of rectangles on a pixel map. And I got to play around with that some more, later on in Squeak, where I did all the color support—see, this was still all black-and-white graphics. But [in Squeak], I put in all the color support and different color depths, and then I added the ability to do warping and rotations.

Hsu: So let's talk about Smalltalk-76. I guess sort of emblematic of that period, you and Alan were sort of diverging where there was a famous offsite that the group went to at Pajaro Dunes [ph?], where Alan said he wanted to burn your disk packs and start all over. And he was sort of looking at moving on to the Notetaker. And the rest of you weren't. Could you sort of talk about that kind of—like how that sort of turn felt at that point in time?

Ingalls: Yeah.

Hsu: Sort of what are the driving forces that sort of where Alan and the rest of the group were kind of not on the same page anymore.

Ingalls: Well, so it's hard for me to tell about all the rest of the group, because that would include. I had some things going on. I have a little bit of trouble remembering exactly when that—when the "burn the disk packs" meeting was with regards to Smalltalk-76. Once I was into doing Smalltalk-76, nothing could have gotten in my way, and it may have been during that period. Because I would have—I wouldn't have been aggressive about it, but I wouldn't have changed my direction. Because at that point, it was so clear to me how to do Smalltalk-76, and that it would be remarkably faster and better as a piece of computer science. So that was, for me, what was going on. With the rest of the people, I'm not sure. Alan did have some other ideas going on, and I don't know if they—he says they were about the Notetaker, so that's probably the case, and he may have had a different software design involved for that. I know that when we got Smalltalk-76 running, there was—on the day that we got the first piece of it actually running, Alan came into the office and he said, "I've got this great new scheme!" <laughter> And we all just looked at each other and said, "What??" <laughter> And that was something that never actually got built, but I know that at first, he did not like Smalltalk-76, because it was a serious language, and it was not as easy, it would not be as easy to teach to kids as Smalltalk-72. And one of the things I failed to do at that point, and it would not have been hard, is to do what I've done since then, which is to implement Smalltalk-72 in it. Because Smalltalk-76 could have run Smalltalk-72 fast enough for the kids. And then we would have had—sort of had our cake and eat it too. But then, Alan, also, I think had other directions going on in his life at that point.

Hsu: Oh, right, yeah. But the–

Ingalls: Since that time, he's had a chance to play with Smalltalk-76 [and its variant Smaltalk-78], when we recreated it in a browser, and he likes it! <laughter> I'm glad to say. <laughter>

Hsu: But I mean, you know, we were talking earlier about the things that were, you know, while you were already—while work on 74 was already going on, the—its ideas that would become part of 76 were already percolating. And so the direction had already been set earlier on, and so it wasn't as much that by that point, you know, you had already been working on these things, this whole time, was the case?

Ingalls: Yeah, although it's true that some of the direction—so for instance, making classes be first class objects and having an object to represent stack frames. That thinking was going on. But it was still, the Smalltalk-72/74 world was just way too interpretive. And therefore slow. And when the Smalltalk-76 design came to me, I could see all of a sudden, everything was going to be incomparably faster and cleaner. So that made a watershed, and that's why, as I say, I didn't do much with Smalltalk-74, except to build an emulator for Smalltalk-76, so that we could start to build it. One of the things that we did in the group from the get-go is to use our systems to build the next system. I'm still doing that, you know? Because they were such productive environments, and so easy to go meta and have specially tailored viewers for this aspect or that, so you could tune things, that kind of thing.

Hsu: I think you've talked about in some of the papers that you wrote, that it was, the group was very much working like running an experiment, trying something, implementing it, and then using that through the feedback to sort of do the next system, and iteratively going on and on.

Ingalls: Yeah, I think I made the analogy to the scientific method, which is where you build something, you make an observation, you get a new theory, and then you build a new thing based on that new theory. And it's the same thing with a system that people are using, which is if you build a system, you let them use it for a while, you find out what's good or bad about it, and you design the next system with that incorporated and then you go around the loop.

Hsu: And it also sounds remarkably like Doug Engelbart's notion of bootstrapping, using the previous system to build the next thing.

Ingalls: Yep, we totally did that, yep. <laughter> Yep.

Hsu: So you were talking about the key thing about 76 that you wanted to make sure that it was fast, and it wasn't as interpreted. It sounds like the byte code part of it was a critical piece to making that happen?

Ingalls: Well, all sorts of things. So the first challenge was to figure out how to make a language that had the same feel as Smalltalk-72 be compilable, because Smalltalk-72 was not compilable, as we've discussed. And so I figured out how to do that with a keyword syntax that would give us that same feeling that we got from the noise words. So the parameters could be named. And then a way to interpret those. And my first goal was to get something compilable. Then the second goal was to make it run fast. You get that from the compilation. The third goal was to keep it small. We had a “small is beautiful” sort of philosophy from the beginning. I showed you that the bootstrap for Smalltalk-72 is 38K including comments. And Smalltalk-76 was also small. So the byte codes gave you that compactness. So you get compactness, speed and compilability, which meant readability. You know, we've talked about how generally you could read most Smalltalk-72 programs, but rigorously, you had to look at how things were implemented to know exactly what the code meant. In Smalltalk-76 it was all deterministic.

Hsu: Right. Hm. What were the other major components of Smalltalk-76 that we haven't discussed yet?

Ingalls: Well, one of the neatest things was that classes became first-class [objects], and then—and this was kind of revolutionary, although there was the hint of it in Smalltalk-74—[even the] actual stack frames, which we called contexts, were real objects. Which you'd think would be a huge amount of overhead. I mean, just to call a procedure, you've got to create an entire object for its stack frame. But the engine was designed so that allocating and deallocating things was very fast. There wasn't a lot of overhead, but the beautiful thing about it was that if you had a bug and got stuck, you could write a debugger completely in the host language. In most systems, the debugger is a very complicated piece of code that has to mess around with [primitive structures]. But this is an object oriented system, and the stack frames are just objects, and you can talk to them with Smalltalk code. So we could actually suspend the project—the process over here, and have some other running Smalltalk code that's just doing what a debugger does. And I show that in the Smalltalk-76 paper. And there's actually code in the system where you could send the message “step” to the context, and it would do exactly what the actual machine did when it was running. And you could step through code very simply just with that. And you could read the program for how it worked, and it was [actually] documented [this way], it was a reference interpreter. So that was a real forward leap.

Hsu: Yeah!

Ingalls: We also had inheritance. And inheritance works beautifully together with polymorphism. So polymorphism means that—at least in Smalltalk, it means that a piece of code can operate with many different data types, for instance. And I can show you the example with Max, but if you take the procedure Max, which gives you back the maximum of two integers, or two numbers, there's code for that in Smalltalk. And it's a very small amount of code, it's like 20 bytes of byte code. But the remarkable thing is that it's described in the Magnitude class, which is the superclass of integers, floating point numbers, extended precision integers, dates, times, all of these objects can use the exact same piece of code, because all that code is doing is sending messages, "less than" and "greater than." And so it's the polymorphism of this code allows that same piece of code to be used in 20 different places. So that allows the system to be much more compact, and always when you make things more compact, more factored, they're more malleable. So if you need to change something, you only have to change one thing, and so those are the main forward steps with Smalltalk-76.

Hsu: Right. So and earlier we had talked about the reason why you hadn't done inheritance in Smalltalk-72. Why was it now important to do that in 76?

Ingalls: Well, so 76 was serious software. You know, you could write big systems in it. [In Smalltalk-72] we didn't do much calculation with dates; we didn't do extended precision integers; we didn't have a date class and a time class. But those are examples of things that all are subclasses of magnitudes. And then there were—we had, the whole file system was written that way, and we did database work. Once you

start trying to build [complicated] systems in it, not educational examples for kids, then you do find places where inheritance is useful.

Hsu: Because then it allows you to reuse code?

Ingalls: Yep, yep. Exactly. And so you may have—we would have sort of skeleton parts of applications that people could take and then build their own part to it, without having to rewrite all the code, but just inherit from them.

Hsu: Right, yeah. Oh, so I guess the decision to make Smalltalk-76 a serious language was a major departure from Alan's vision. How did the group come to that conclusion that you guys wanted to take Smalltalk in that direction?

Ingalls: Well, I can't remember how it happened in the group, but I know how it happened for me, which is that as soon as I saw that this could be done efficiently, nothing could stop me. I mean, I would have gone somewhere else and done it, and I think everybody was sort of enthusiastic about it. How could they not be? <laughs>

Hsu: Right. You were the programmer, so–

Ingalls: Yeah, I have a practical streak, and Adele has a sort of a commercial or industrial streak along with her educational side. And I think she could see that it would be good to have something that was, you know, serious [or even saleable], software. And so I think she kind of went along with that. And in fact, it was she who then got excited about commercializing Smalltalk-80. And plus we had the kids covered with Smalltalk-72 which we continued to use it for quite a while for teaching.

Hsu: Let's see. Could you talk about some of the other pieces that went into–

Ingalls: I just realized this thing is going to go off, so I'm just going to—because I started it up during lunch. Sorry about that.

Hsu: Could you talk about the contributions of other people that made it into Smalltalk? Like Ted Kaehler's work, Dave Robson, Peter Deutsch, Diana Merry, Larry Tesler. All those—so much work ended up going into Smalltalk.

Ingalls: So Ted worked—well, I talked about him having worked with the line drawing for the Turtles. And then he also worked with me on the storage manager for Smalltalk-72. And then he and I did a crazy

virtual memory called OOZE that served Smalltalk-74 and Smalltalk-76. And it was really important, because those machines were really small machines. So the Alto was—some of our Altos were 48K. It's 16-bit word, so it's 96K bytes. But that's a small machine to do anything serious in. Plus you have to subtract out the allocation for the interpreter, and the display. So there's very little room left. And OOZE allowed us to take—to in one 16-bit pointer, talk about the entire size of our disk packs. So basically everything we could put there. And we had a scheme for swapping things in and out. And it was much more space efficient than a paging system. So that was great. That's another thing that Ted did. And he had some unique contributions to that, too. Diana Merry had worked primarily on the text display, and she carried that on even after BitBlt was done, the new way, she worked on the paragraph editor and a sort of a document editor that we made. There was Steve Weyer who came and he came with—I think he had been with Adele, I'm not sure. But he did an interesting information retrieval system that he designed together with Alan called FindIt. And it was actually used to manage the Xerox library, I believe. That's Ted and Diana. And who else have we got there?

Hsu: Let's see, Larry.

Ingalls: Larry Tesler, as soon as we got Smalltalk-76 going, he had a design for a multi-pane browser. And he really—the whole paned window metaphor that we also used in the debugger and stuff was his original design. And he did some other interesting work, using Smalltalk, but it wasn't necessarily a part of our project. They did an interesting text editor, and then he did a separate text editor not in Smalltalk at Xerox, too. And he had a design for a small, very simple version of Smalltalk to run on the Notetaker that was separate from the version that we did. We did a port of Smalltalk-76, which was called Smalltalk-78 that ran on the Notetaker.

Hsu: Oh.

Ingalls: We can talk about that sometime.

Hsu: Yeah, but Larry's was different than that?

Ingalls: Yeah, he did a small thing as sort of a fun project with Kim McCall. And they got it running. And it was a little bit different. But that's–

Hsu: Let's see. Are there any other people that I forgot to mention?

Ingalls: Yeah, I think I've left out one of the list that you mentioned.

Hsu: Let's see, Dave Robson?

Ingalls: Oh, yeah, Dave Robson! So Dave worked—he worked a bit on Smalltalk-72, but the project that sort of he then took on largely by himself—I mean, we all worked together—was the Smalltalk-74. So he did the—got the making messages be a real—making messages be real objects. So that some of the primitive message operations in Smalltalk-72 could be described in terms of Smalltalk itself, rather than being primitive.

Hsu: Yeah, and Peter Deutsch had been part of the Computer Science Lab.

Ingalls: Right.

Hsu: At what point did he jump over?

Ingalls: So Peter came over somewhat later on. I'm pretty sure—well, I know that Smalltalk-76 was all running at that point.

Hsu: Oh, okay.

Ingalls: And we were probably actually even starting on Smalltalk-80. And we were—that was the time at which we were getting ready to run Smalltalk-80 on some of the other Xerox machines, including the Alto. Excuse me, including the Dorado. And one of the other ones, the Dolphin, they called it.

Hsu: Yeah.

Ingalls: Okay. And so Peter did a high performance implementation, just essentially doing the virtual machine on the Dorado. And then he got interested in pursuing that, doing a commercializable version on the Sun Workstation. And he worked on that together with Dave Ungar and [Allan Schiffman]. Yeah. <laughter>

Hsu: Okay. Wait, you mentioned Steve Purcell earlier, as well.

Ingalls: Oh, yeah, so Steve Purcell came on with us pretty early. And he was really sharp and he did a fair amount of microcode work for . So he had a neat little animation kernel. And Ted did the high-level version of the music editing. And the high-level version of the animation system was [Ron Baeker and Tom Horseley].

Hsu: Okay. <laughs>

Ingalls: But yeah. So he did those microcode pieces, and then he left to go elsewhere. And Bob Shur worked on [the microcode for music synthesis.]

Hsu: And also John Shoch was originally part of the Learning Research Group.

Ingalls: Yes. And when he came in, I'm not exactly sure. But it must have been fairly early, because—and I think he worked somewhat more with Adele, and he did write that little code editor that [I mentioned].

Hsu: Right.

Ingalls: Yep.

Hsu: All right. So then let's talk a little bit about the Notetaker and Smalltalk-78.

Ingalls: Okay.

Hsu: So at what point did that start to happen?

Ingalls: Well, so I don't know exactly when and how it started, but Alan wanted to do a portable machine, we all wanted to do a portable machine. And Doug Fairbairn took it on as a project to build it. And I think we had sort of hoped it would be more portable than it wound up being. It wound up being [a suitcase] scale [similar] to the Osborne machine, if you're aware of that. I would call it "luggable." That's what we called it. And the principle behind it being a Smalltalk machine was that I figured that I could do a Smalltalk that would run on an 8086. So this was right about the time that Intel had just come out with the 8086, which is the first microprocessor that I thought was powerful enough to run our software. Could have run on the 68000, also, but that was our choice. And the—so that would mean rewriting all of our Smalltalk code for the Intel processor. And we had a lot of graphics code involved in all of the line drawing and stuff. If I could get BitBlt to run on the Notetaker, then there was a whole lot of [other graphics] code that we had that I didn't need to rewrite. I could write that all in terms of Smalltalk calling BitBlt. So that's what I did. And as a result, the entire interpreter was, I think 6K of [8086] code. [A huge reduction.] And so we got it running, and it ran Smalltalk as well as the Alto. So that was good. [The NoteTaker] didn't get used much, we took it to do several demos. But there wasn't a commercial direction for it to go yet. But it was a nice design. It had a touchscreen, 640x480, with full bitmapped graphics on it, so you could do a lot of things with it. And it's funny, it had an Ethernet processor on it, and another file processor. The

Ethernet processor was a specially selected 8086 that ran faster. And so I would move code into the Ethernet processor to make it run faster when we didn't need the Ethernet. <laughter> But–

Hsu: That's a fun hack.

Ingalls: Yep.

Hsu: At what point did Alan decide to leave the group?

Ingalls: I'm not great about time. It was right around then.

Hsu: So like, the work on the Notetaker was already taking place.

Ingalls: Mm hm.

Hsu: And he went on sabbatical?

Ingalls: Yeah, and then I guess then went from there to Atari, right?

Hsu: Yeah, I think so.

Ingalls: Yes. And there were other things going on in his life at that point, too.

Hsu: Right, yeah. So what was it like for the group for Alan to leave?

Ingalls: Well, at that point, we had decided that we wanted to somehow get Smalltalk out. Because it was significant. Everybody who was serious on the outside about software said, "This stuff is great!" And Adele wanted to at least get it out. And I think she was maybe thinking of commercializing it. So that led to the move from Smalltalk-76 to Smalltalk-80. And the move—that move involved a couple of things. One of them was standardizing the character set, because we had a bunch of strange characters as part of the Smalltalk-76 system that just made things look somehow more readable, we thought. And but we had to go to a ASCII character set. The system needed to be better documented and Adele took that on, as a responsibility and did a great job with that. And then inevitably we did a few more things with the system that made it a little bit nicer, and I can talk about those. So those all work together as a package of a new Smalltalk, and one that could be published to the world. And so with Alan's absence, we sort of gravitated into that export publication kind of [mode], and that's why the Smalltalk icon is a balloon. I don't

know if you know the story of the balloon, but basically there had been an issue of *Byte Magazine* that showed Pascal as an island, well, sort of the Sea of Pascal, and then Smalltalk was represented in it as being on the Isle of Smalltalk, isolated from everybody else. And this bugged me, and I think bugged everybody else, and we wanted to somehow not be isolated. And I had one of my childhood loves was the book, "The Mysterious Island." And it's about an escape that some people make from behind the Confederate lines in a hot air balloon. So my picture was that our export of Smalltalk from the island could be by a hot air balloon. So that became the cover of the *Byte* issue that covered the release of Smalltalk-80.

One thing I'll say about the character set is—well, it's a detail—but we lost one nice thing about the syntax in Smalltalk-76, which was the terminal left arrow for assignment. And we could get into that about the message patterns there.

But Smalltalk-80 introduced Booleans as being a class. They were not a class in Smalltalk-76. And this made Smalltalk be almost unique in being a system that you could—almost every other system, if you ask how "if" works, how conditionals work, you trace it down to the bottom, and still there's a conditional there. there isn't anything that—it comes down to machine code doing a test or something. So the beautiful thing about message passing is that the true and false in Smalltalk is done simply by sending a message. So if you make a test, you say, "If A is greater than B, if that's true, then do thus-and-such." Well, that whole "if true" is a message sent to the Boolean and if you go and look in the class for true, it does one thing, and in the class for false, it does another. So there's never a test that actually happens, it's just the mere fact of sending the message to the right receiver takes care of that. So it's kind of a unique feature of the language that conditionals are able to be represented without needing conditionals. So. <laughter> And other than that, Smalltalk-80 is pretty much Smalltalk-76 commercialized.

Hsu: So was that—because Adele wanted to make this push for commercialization, was that how she basically became the de facto manager of the group?

Ingalls: Yeah, so she—when Alan left, that became our project, which was to get Smalltalk out. And Adele took that on as her responsibility. I was happy to have that happen. And also the documentation. And the writing of the books. And I actually participated a lot in writing of the books, but it became almost an obsession toward the end about how much energy went into the Blue book, and I finally said, "I'm going to resign from the authorship list. I know I've done a lot, but I need to focus elsewhere." So that's why I'm not on the Blue book authorship. But I did—I contributed heavily to the Green book, which talks about all the implementation and stuff.

Hsu: Right.

Ingalls: Yeah, so we kind of all gravitated to that, and it was set up then that it would be—we'd get some other groups in other companies to do reviews, essentially to receive the—to get a free license to try it out, and to port it to other systems.

Hsu: Right, because you needed implementations, other implementations.

Ingalls: Right. So we had a group at Tektronics, a group at DEC, and a group at Apple. And is that it? I think there's one other, but I can't remember.

Hsu: So what was the advantage of having the third party involvement?

Ingalls: Well, for one thing, it put it out in the world where a lot more people could see it and play with it. It didn't get a lot of momentum in any of the areas except for Tektronics, and Tektronics got serious about it. They were a group that recognized the interesting aspects. They were also—they had a workstation built around the 68000, I think, that they were selling into the—they started selling into the high-end workstation market. And Smalltalk seemed like a good fit for that. And they also actually did some projects with doing Smalltalk for user interfaces in some of their oscilloscopes, the high-end ones.

Hsu: Could you talk a little bit more about publishing the *Byte* articles and the books?

Ingalls: I could say one more thing about the Tektronics connection was that one of the people up there was Ward Cunningham.

Hsu: Uh huh, oh!

Ingalls: And he started wikis.

Hsu: Yeah.

Ingalls: [After the release of Squeak] there was a very simple way of doing wikis and servers in Smalltalk. So there was a little while when people would [download Squeak] just to get a wiki going.

Hsu: Oh, wow! <laughs> That's a cool story.

Ingalls: Yep.

Hsu: So you mentioned you did a lot of work for the Green book and the Blue book, and you wrote those two articles in the *Byte* Special Issue.

Ingalls: Right.

Hsu: How much of the work of the group was doing these publications versus working on Smalltalk itself and other things?

Ingalls: So most of us were not particularly oriented to publishing or giving talks, you know? I did—I wrote my paper on Smalltalk-76 and presented it. Adele was the main force behind publication and documentation and stuff. So that was really useful. And she kind of organized a group around that. And we weren't resistant, it's just that that was not our—we were more doers than talkers. And that also came from—I mean, until he left, Alan had been the major voice of the group, and we would all fall in behind him and do everything that needed to happen, even if it looked like it couldn't be done. Another organizational activity at Xerox that Adele spearheaded was a big communication demonstration we did for Xerox executives. And she worked on setting up a simulation environment in Smalltalk where it simulated work processes. And there were, I think, eight versions of it for each of the eight executives coming. And we got paired up with—one of the Smalltalk workers together with an executive—for a day-long demo. And in the process of that, because Smalltalk was nice and high level, and Adele had set up this sort of curriculum for it, every one of those executives got an experience of what it was like to work with software, and I think it was all a great success. And certainly gave our group, which was sometimes viewed as the renegade group, a sort of a good corporate responsibility panache.

Hsu: Do you remember which thing that you worked on? Which demo for which executive?

Ingalls: Yeah, it wasn't Kearns. No, I think I had Goldman.

Hsu: Oh, wow.

Ingalls: Yeah.

Hsu: Oh <laughs>, and what was the actual simulation?

Ingalls: So they were, in each case, there were entities that were producing things and entities that were consuming them, and then there were queueing situations. And you could go and you could modify the logic of the queues and get it so that things would work well. I don't remember much of the details of it, but it was nicely scripted. It was a good piece of curriculum. That's what Adele was good at.

Hsu: But Jack Goldman was already sort of a PARC, I mean, he was the head of—in a way, the head of the research arm. So he was already kind of bought into what PARC was doing.

Ingalls: That's true. Yep, yep.

Hsu: So I guess it wasn't as much of a sell for him, necessarily as it would have been for one of the other executives.

Ingalls: Yeah, that's true, yep, yep.

Hsu: Well, while we're talking about demos, were you guys involved in the Futures Day demonstration in Florida?

Ingalls: I wasn't. I think we prepared some of the demos, but I didn't go on that. And I can't remember exactly who did.

Hsu: Okay. Do you remember what kinds of demos you prepared?

Ingalls: No. Nope, sorry about that.

Hsu: Yeah. And I think Adele was saying that the group was giving demos all the time to people outside of Xerox. Sort of what kinds of people were coming in to see this?

Ingalls: Some were Xerox executives wanting to understand what was going [on] at PARC, and then we would have visits from people in the computer science community, who were interested in—they had heard about Smalltalk, and were curious about it. Interested in graphics, interested in sort of live object systems like ours.

Hsu: So people from like Stanford or other universities?

Ingalls: Mm hm, yep.

Hsu: Yeah.

Ingalls: Yeah, people from universities. And then there were other commercial groups within Xerox, like the people who were building the workstations.

Hsu: Like the Star group?

Ingalls: The Star group, yeah. And before that was the Dolphin, which never really got commercialized, but—and we would go to—I can remember a couple of shows where they were showing the hardware, and we'd show Smalltalk on it. Because our demos were more interesting than some. <laughter>

Hsu: So doing those demos so frequently, I guess you had—did you sort of come up with a kind of a script by a certain point that you would always kind of show a certain kind of a thing?

Ingalls: Yeah, we would—so there was—there were graphics things we could show, so we had some nice drawing tools using the nice aspects of the bitmap displays. We had a document editor, which—where you could mix text, media and graphics—that was revolutionary at the time. And then for programming, I would give a Smalltalk demonstration, and Smalltalk could do a lot of things that you just couldn't do in other languages. We'd put up something with an error in it, and put the debugger on the same screen, fix the error and proceed from there. And things that were not [possible] to do in other systems.

Hsu: Okay! So let's talk about probably the most famous demo <laughs>, which I've been dancing around up till now. So that's, of course, the Apple ones. From your perspective, how did that come about? I've heard this story, of course, from Alan and from Adele. But I'd like to see sort of from your perspec– like where you were standing, how did that come about?

Ingalls: Okay. Well, so I hadn't been involved in setting it up at all. I was glad to know that—I had always felt positive about Apple, and I kind of hoped that we might do something together. But as it came to me, there was going to be a visit from Steve Jobs, and some other people, and that I would be one of the people [giving] the demos. I think Larry Tesler was another one. And so I just—it came my chance to give a demo, and I did one, and the particular thing that I showed was I showed all of the sort of programming and debugging features. And then I showed that while in the process of selecting text, I could interrupt the process, go and change how text selection worked, and continue from there with the selection working differently. And that was about as extreme a thing as you could show for malleability of software. And I think that played pretty well. And then Steve came back with something to me, which is, he said, "I notice that your text scrolls a line at a time. Could you make it be a continuous scroll?" And I said, "That could take a little more work, but I could show you how to do it, but I don't know if you have time." And so they were about to take a lunch break. And so they took a lunch break, and during the lunch break, I got it done. It was not that hard, but I basically had to find a place where we'd go up [a] line at a time, and I broke it down into a pixel at a time. And so when they came back, I showed them that done, too. And it

was a great thing for Steve to ask for, and it really looked nice. So that was the extent of my involvement in the demo. And it was really fun, too, because, well, for one thing, he was enthusiastic, and I knew he got it, and sometimes you don't always get people who get it from a computer sort of side, but also have that feel of entrepreneurialism around them. So that made me enthusiastic about it. I had always felt that what we did at Xerox was—I don't know how to say it, but it's the kind of stuff we did, all you had to do was see it. [There isn't] a secret to be kept. All you have to do is see something done that way, and then you can go off and figure out [any other way] to do that same thing. And so I think that a lot of the spirit of what we did was transmitted in that meeting.

Hsu: I heard that there were actually two separate meetings. Do you remember–

Ingalls: I only was in one, and what the other one was I wasn't a part of.

Hsu: Okay, right. Do you remember interacting with some of the other people that had—from Apple that were there?

Ingalls: That came on the visit? Not really, no. And so I got to know Bill Atkinson later, but I didn't really know him then. And I'm not—who else came along? I'm not sure. But the answer is that I didn't really—there wasn't a chance really for me to talk with them. It wasn't a sit-down meeting. It was a demo.

Hsu: Right, okay, yeah. We talked about this a little bit over the break. But one of the things that—and of course, you just mentioned that you showed them what *could* be done, and then they went off and sort of did something similar. Bill tells this famous story about how he thought that you were clipping regions in order to save processing time, and you weren't actually doing that, but he thought you were doing it. And so he went and implemented it. <laughter>

Ingalls: Yeah. Yeah, that's pretty funny. Merely the way that we worked our paned windows together, with how we worked BitBlt made it look that way. But we weren't, yeah, and it's interesting that he did that. He did a great whole package of general regions with—especially with curved regions as part of his QuickDraw work.

Hsu: Right. So what did you—what were you thinking about whether Xerox should be giving these demos, because Adele was not so super happy about the fact that you guys were giving these demos. What was your take on it at that time?

Ingalls: Well, philosophically, I've always wanted to give away everything I did. <laughter> I mean, I also want to make money <laughter>, but no, I was glad to see things go that way, and I wound up going to Apple just hopefully to get Smalltalk to go out more widely. That didn't really happen, but there came a

time when I left Xerox, I went to Apple, and I picked up some of the work that had been done. Apple had been one of the reviewers of the Smalltalk-80 release, but it hadn't really gone anywhere. And so I picked it up when I got there, and did a version that was distributed through the Apple Programmers and Developers Association. And it became a really usable Smalltalk that was fairly well supported on the Mac platform, and it was nearly free. So that was nice.

Hsu: Yeah. You had mentioned that—I mean, this was around the time when microcomputers were starting to come up and you said you were very supportive of what Apple was doing, even while you were at Xerox. How did you get involved in sort of that microcomputer scene? What was your view of that, compared to what Xerox was doing?

Ingalls: Oh, I thought it was great! So and I worked really hard to make the Notetaker Smalltalk work. And the Notetaker was kind of big and clunky. But I could see that things were going that way, and the fact that we got it to perform actually better on the 8086 than it did on the Alto, it was clear to me that we were making progress. And it ran decently on the Mac also, which was, the early ones just the 68000 and then there came to be more powerful ones.

Hsu: Right.

Ingalls: By the time that I was at Apple working with the 68000s, we were using the 68020s and large displays, and that was a really productive environment.

Hsu: Mm hm, yeah. And you told me a little bit before, during the break, that you had an Epson portable? Was that here or–

Ingalls: No, that was from a long time before.

Hsu: Okay. So that was in the '70s?

Ingalls: Yeah, I think maybe late '60s.

Hsu: Oh, really?!

Ingalls: So, yeah.

Hsu: Oh, so this is way before PCs.

Ingalls: Way—yeah, this is way back. Sort of the first of the somewhat small machines. Yeah, that's another thread.

Hsu: So what was your first actual microcomputer, or PC? What would you consider a PC?

Ingalls: Oh, well, that would be it.

Hsu: That would be it. <laughs>

Ingalls: Yep. So they had the Epson HX-20. Well, what was cool about it was that in this one package, it had both a microtape file system, and an actual printer, and a bitmap display, and it ran BASIC, so it sort of had all the components. They were all tiny, and not that powerful, but you could do lots of fun experiments with it. So that was—yeah.

Hsu: So you had that machine before you even went to Xerox.

Ingalls: Yeah, yep.

Hsu: Okay. <laughs> Okay, so then, so we already talked about you moving to Apple. What ultimately made you decide to join Apple and leave Xerox?

Ingalls: Well, let's see. I went to a conference, and Larry Tesler was there, and Alan had moved to Apple at that point. And he was there. And I had lunch with them, and they said, "Gee, it'd be nice if you were here and you could do good things." So and that sounded good to me at the time. I kind of had finished up what I was doing at Xerox at that point. And I had wanted to, as I said, get [Smalltalk] running well on the Mac, and get it out in the world. So at that point, I moved over.

Hsu: And so that was what year?

Ingalls: Oh, gosh, that would have been—I think it was about '84 or '85. It was [just] after the famous Mac opening, which was '84. So it [would] have been late in '84 or early '85. <inaudible> Yep.

Hsu: And so then you joined—did you join the Advanced Technology Group?

Ingalls: Mm hm, yep, yep. And I worked specifically on Smalltalk there. And I worked on a project—well, I just got the Smalltalk release ready, and then I worked on a system called Fabrik.

Hsu: Oh, okay.

Ingalls: Fabrik was a visual programming language. Basically, it was a dataflow system, so you could piece together pieces of a user interface, like Lists and Text, and assemble the user interface you wanted, but they were all components that you could wire from one to another, so you could actually be building the user interface, while you were programming the parts of it. And then the things that you built could be saved and run on any other Smalltalk. And I saw it also, as a program generator, because I saw how you could take what you had done in this way, and for instance, generate a Pascal file if you wanted to do other Apple software. But it was not really picked up by the rest of Apple.

Hsu: So you had earlier mentioned that you had taken—so that Apple had already done some Smalltalk work before, prior to you being there, because of the work on Smalltalk-80. And then your addition was to–

Ingalls: So I did a couple of things. So yeah, Rick Meyers at Apple had done the—he had been the engineer at Apple who sort of received the Xerox test project. And so I did a couple of things. I worked on the memory system so that it was a better garbage collection system. And then I also worked on—oh, what else did I do? But I just—I lost my thread of mind there. Well, that was—I think that was most of it. There was a lot to—and making it work on the bigger Macs.

Hsu: Mm hm. On the 020-based machines?

Ingalls: Yep, and with bigger displays.

Hsu: Mm hm. <inaudible>

Ingalls: That's most of it, yeah.

Hsu: Okay. Actually I forgot to ask you about garbage collection earlier. I think in some of the papers you wrote that you—the original Smalltalk-72 used reference counting–

Ingalls: Mm hm.

Hsu: –as, for memory management. Did that continue or did Smalltalk transition to using a different kind of garbage collector later on?

Ingalls: Yeah.

Hsu: Like a more automatic version.

Ingalls: Well, they're all automatic.

Hsu: Well, right.

Ingalls: But yeah, so the early Smalltalks used reference counting, which the nice thing about reference counting is that if you don't have much space available, you can get by with less available space. And it can be faster. But the problem is that you also—you can get structures that aren't reclaimable, so you need some sort of backup garbage collection. And we did that—in Smalltalk-72, you got that by restarting. <laughter> But for the most part, Smalltalk-72, the things that we did in Smalltalk-72 were small enough that we didn't need full garbage collection. And then when we went to Smalltalk-76, again, it was a big enough space that we didn't. And we did have a way of doing sort of an overnight garbage collection process for when that wasn't adequate. And then essentially every Smalltalk that we did since then, we had either both—or a full suite garbage collector.

Hsu: Mm hm, okay.

Ingalls: Yep, and that's—automatic storage management is a real part of the sort of feel of object oriented programming, because you think of all these objects as being objects by themselves, and they live in—connected to the rest of the system. And you don't want to have to be thinking of how to reclaim them. That just makes the code more complicated. And usually it's wrong anyway. <laughter>

Hsu: Right. So then you mentioned that the Smalltalk work at Apple did not have a large impact. Was there interaction with—I think Larry mentioned that—I think there was some interest in—or some crosstalk between the MacApp group, because Larry had done MacApp, and the Smalltalk group. What sorts of collab—or I don't know collaborations, but maybe communications were going on between those two?

Ingalls: We didn't do much with the MacApp group. The one thing that I mentioned with Fabrik being able to possibly generate Pascal sources, I thought that if they were interested, it would not have been difficult to actually have Fabrik in Smalltalk be GUI-builder for Mac App applications. But MacApp had its own framework for doing that, so nothing really went forward with that.

Hsu: Right, but that's interesting that you could have something that was native Smalltalk, but generate Object Pascal code?

Ingalls: Mm hm, yep. Well, that's just because the way the Fabrik framework was set up, it was a known set of widgets with a known set of interfaces, and so you can imagine having implemented those in Pascal or in Smalltalk.

Hsu: Mm hm, okay, I see. Hm. Was there any connection between the Smalltalk work and Dylan later on?

Ingalls: No.

Hsu: No.

Ingalls: Nope.

Hsu: So that was a completely independent.

Ingalls: Yeah, and I don't know, Dylan came on later, I think than when I left.

Hsu: Okay.

Ingalls: I believe. I left in '87.

Hsu: Okay. And Alan was there also. Obviously, you mentioned that he was part of what convinced you to go. Did you actually work with Alan at all while at Apple?

Ingalls: No, not really. No, I was pretty much on my own there.

Hsu: Okay.

Ingalls: Yep. Yep.

Hsu: And so then you mentioned that you left Apple in '87. Was that the point at which you decided to take a break from the computer industry?

Ingalls: Yes, yep. Yep, from then until about '93 or '4.

<break in recording session>

Hsu: So we were talking about your break in the industry.

Ingalls: Right.

Hsu: So you went back East during that period of time?

Ingalls: Yep.

Hsu: And what led you to that decision?

Ingalls: Oh, well, it was basically a family decision. So there was a family business, it's a hotel in Virginia, and there was nobody else in the family to help take over with it for a while. So I went back to do that.

Hsu: Okay.

Ingalls: And I was ready for—I guess I was ready for a break. <laughter> I hadn't thought about it that way. But it was certainly a refreshing break.

Hsu: Yeah.

Ingalls: And it's interesting, you know, when I came back, what, about eight years later, something like that, it seemed like all the computer hardware was running a thousand times faster and a hundred times bigger, and it didn't seem to me that that much had changed in software. So I was able to pick up almost where I left off with a lot of the same software. But running in a much more exciting way.

Hsu: Right. What made you decide to come back?

Ingalls: Well, things came to—things got to where they needed to go to with the family business. And Alan was looking to do sort of a new generation of his work, and was looking around for a malleable system. But actually, I didn't connect up with Alan until a little bit later after when I got back. So yeah.

Hsu: So where were you–

Ingalls: The first place I went to when I came back was Interval Research.

Hsu: Okay, right.

Ingalls: And I worked briefly with Dick Shoup there.

Hsu: Oh, right.

Ingalls: Sort of box language. And then while I was there, I got in touch with the group that was under Glenn Edens. And they wanted a system for doing sort of software for home management—managing a home network. And I convinced them to take a Smalltalk and write a virtual machine in C, and just pick up essentially the Apple work that I had finished with at Apple. And they did that.

Hsu: At Interval?

Ingalls: Mm hm. So Roger Mike worked—I think did the main work on the interpreter. And I kind of coached him along, and I knew all the formats for how to, you know, what was needed. And the Apple version of Smalltalk had no license. It was a completely open license, so it was good for them to use, completely open. And then, at that point, I was starting to talk to Alan and his group again. And I remember having this—I was thinking of maybe going to work with them, and I remember having this ah-ha during a rainstorm driving over to Apple, that I could do all the same work that they had done at Interval, just by using the reference interpreter [written in Smalltalk] from the Blue book. Because the machines were fast enough. And so that's what became Squeak.

Hsu: Oh, okay, yeah.

Ingalls: And it was about week after I started at Apple, we had the Squeak project underway.

Hsu: Okay, so you rejoined Apple at that point?

Ingalls: I did, yep. Came back in Alan's group.

Hsu: And Alan was still at Apple in '93?

Ingalls: Yeah, yeah.

Hsu: Okay, right, okay. Huh.

Ingalls: He had been doing other projects while I was gone. I think they worked on the Vivarium.

Hsu: Okay. Vivarium.

Ingalls: Which I was not a part of.

Hsu: So the Squeak project was started at Apple.

Ingalls: Yes.

Hsu: Oh, okay. Because it sounds like, the branding of Squeak makes it sound like it was a Disney thing.

Ingalls: Well, so, no, but it went—no, it was called Squeak at Apple.

Hsu: Oh, it was even called Squeak at Apple.

Ingalls: Yep, yep.

Hsu: Huh!

Ingalls: Because when I came back, I think there was a little bit of—there was a little bit of conversation about, "Oh, no, not more Smalltalk." <laughter> And I think Alan suggested Squeak as the term for it. And whether he was already imagining a connection with the mouse or not, I don't know.

Hsu: Ohhh! <laughter> But it was just sort of a way to rebrand it.

Ingalls: Yes, yep, yes.

Hsu: But it's the same technology essentially, just a next direction.

Ingalls: Yep, yeah. Squeak is Smalltalk-80 with a bunch of things done in a much neater way. The remarkable thing about Squeak is that it's a full Smalltalk-80 where the entire virtual machine was written

in Smalltalk itself. Which made it metacircular. And we—so, well, I'll tell ya, the way that we did this was, the reference interpreter for the Smalltalk-80 virtual machine was written in a flavor of Smalltalk that's—it uses all the Smalltalk syntax, but it's all done without really needing to send messages. So it can be—so it could be transliterated to machine code, and that's how the various reviewers made their implementations.

Hsu: Oh, okay. Like Tektronics and the other companies.

Ingalls: Right, right. And what I realized was things were running fast enough that we could actually work with it without running it actually in Smalltalk, and that took a little bit of work to do, but we got that running in about a month. And we ran it, I think, in the VisualWorks Smalltalk, which is a good, efficient one as a start. And then right away, I got John Maloney started [on] doing a translator from Smalltalk to C, which meant that we could generate a virtual machine in C that would run at production speed. And got Scott Wallace and Ted Kaehler working on the—Scott worked on the file system, and Ted worked on everything else. I think he did storage management with me once again. <laughter> And the storage manager in Squeak is really nice. So in the course of four months, I think we made the changeover to where John had a C version running. And, oh, and then I worked on BitBlt and I got BitBlt working with all the different color depths and so we just, you know, it really had a different feel to it. So that's how Squeak began, and by the end of the summer, we released Squeak in October or something of '96.

Hsu: Oh, wow.

Ingalls: Yeah. And so it was less than a year from start to finish with that, and it was radically portable because it included its own virtual machine and so I think that Ian Piumarta got it running in—so we just released it for the Mac and it took Ian Piumarta just a week or two to get it running in Linux and then Andreas Raab got it running on Windows in another week or two.

Hsu: Wow.

Ingalls: Which is pretty good portability. And it was written, a lot of the storage mechanisms in it, are written to be [totally portable]—you could [move] it from one system to another whether it was a reverse Endian machine or not, and we paid a lot of attention to portability and all the bitmap graphics were bitwise compatible from one machine to another. So you could have been working in Squeak on a Mac, run into a bug, save your system with the bug in process, and somebody else could pick it up on an Intel processor and pick it up right where you were, fix the bug and be going again.

Hsu: Wow.

Ingalls: So that made it a great system for academia and portability.

Hsu: Yeah. So did Squeak get a lot of take-up in academia?

Ingalls: Yeah, a fair amount. Right. And actually, now I'm thinking that what I told you about the wikis, that may have been in Squeak.

Hsu: Oh.

Ingalls: That that took place.

Hsu: Right.

Ingalls: So it was a little bit later from then. Yeah. There are a bunch of universities and colleges that still teach Smalltalk for its various good properties, and most of them use Squeak for that. One of the really nice things about Squeak that distinguishes it from—we kept it compatible with Smalltalk-80 because that really helped synergy in the community, but the regular Smalltalk-80 interfaces, what's called an MVC interface, and follows the sort of Xerox and, well, their whole tradition. What we did with Squeak was we picked up Morphic from the Self project. John Maloney was part of the group at that point, and he had worked on Morphic with Dave Ungar and Randy Smith at Sun, and so we did an implementation of Morphic in Smalltalk and it made the system really nice and easy to use.

Hsu: What, I mean, you mentioned MVC before. So Morphic and MVC are kind of different approaches to solving the same problem? Is that accurate?

Ingalls: Yeah, I would call them different user interface paradigms.

Hsu: Okay.

Ingalls: Yeah.

Hsu: Right.

Ingalls: And I would say that MVC is, at least the Xerox MVC, that continued on, is mainly tailored towards kind of paned-window type applications. Morphic is much more general. It's much more easily

adaptable to all sorts of graphical situations. That's not to say you can't do that kind of thing in an MVC system, but you have to build more of it. Morphic is really flexible that way.

Hsu: Right. I see. Hm. At what point did you move to Disney?

Ingalls: So, we moved to Disney very soon after the release of Squeak.

Hsu: Okay. The whole group en masse?

Ingalls: The whole group, yep.

Hsu: <laughs>

Ingalls: Yep. So Alan, I guess, had had an arrangement with Disney or—and it was okay with Apple for us to do that. We made sure that it was before we left.

Hsu: Right.

Ingalls: Yeah.

Hsu: So then the work just continued on at Disney almost–

Ingalls: Yeah, well, I think the thinking behind this was that we were, as always, interested in doing educational software, and that if we could do a good job with educational software, that Disney would be an ideal institution to publish that work and really make it widely available, and it's too bad. We did manage to make some pretty good educational software, which we can talk about, but somehow it never really got picked up in Disney. But what we did was the Etoys environment.

Hsu: Oh, okay.

Ingalls: Which is a really nice environment and it's the forerunner to Scratch, which is huge now.

Hsu: Right.

Ingalls: But it had all the same sort of tile programming capabilities.

Hsu: Right. So it's much more of a visual programming environment. Kind of like Fabrik but not, but more catered towards children?

Ingalls: Yeah. I mean, Fabrik was specifically data flow, and Etoys was not data flow, but Etoys did have the capability of doing programming by moving tiles around, and the nice thing, the two nice things about tile programming, one of them is that you don't have to type, so that helps a lot with younger kids, and the other is that if the tile editor is done right you can't make mistakes in syntax, so that means that there's a whole set of debugging issues that you don't need to get into.

Hsu: Mm-hm. Right. And Etoys was sort of implemented on top of Squeak?

Ingalls: Yes.

Hsu: Okay.

Ingalls: Completely. That's how it was done, yeah.

Hsu: Right.

Ingalls: Yeah. And so I was responsible, for [the overall plan] in Squeak, and there's a [nice] paper about it ["Back to the Future"]. Then I, and the rest of the group, all contributed various things to the Etoys environment.

Hsu: Mm, mm-hm. So then how much more did you, how much did Squeak progress at Disney while you were there?

Ingalls: Oh, gosh. We always had something to do.

<laughter>

Ingalls: So I think it was while I was there that I did—that's probably when I did WarpBlt, which meant that you could, even though it was a bitmapped environment, we could do arbitrary scaling and rotation of things. I remember somebody from, who worked on some other project, saying, "You guys sure got a lot of mileage out of that WarpBlt thing."

<laughter>

Ingalls: Because he had been working with a, you know, a more general graphics environment, but we managed to get a lot of that strength without having to, while still working really simply.

Hsu: Hm. So then at what point did you leave Disney? So it looks like 2001 Alan left Disney.

Ingalls: Mm-hm.

Hsu: So did the entire group also move with him again?

Ingalls: Pretty much everybody except John Maloney.

Hsu: Okay.

Ingalls: Yep. And I didn't go with him at first. Let's see. I consulted for a while at HP with them, and then I just sort of left for a couple of years. I was up in Truckee and I retired sort of, and then, then later on, decided to come out of retirement and went to work at Sun Microsystems.

Hsu: Okay. What were you doing at HP?

Ingalls: Still continuing with some of the Smalltalk work. So what I did there was, this was at the time when people were starting to have 64-bit machines, so we did a whole, another one of these compatibility things with Squeak, which is to make it, we had it so that you could run a 64-bit image on a 32-bit machine or a 32-bit machine on a 64-bit or, you know, you could do all the combinations. So that made it easy for people to develop and cross-develop between those platforms.

Hsu: Right. So you were doing that work as an HP employee but also collaborating with Alan's group at [VPRI].

Ingalls: Yeah.

Hsu: –Viewpoints?

Ingalls: Yeah.

Hsu: Right.

Ingalls: Yeah.

Hsu: Okay.

Ingalls: Yeah.

Hsu: Yeah.

Ingalls: And then we brushed up against the group that was doing E, Mark Miller and his group.

Hsu: Okay.

Ingalls: And I started on a sort of a modular Smalltalk version there but then I dropped that.

Hsu: Right.

Ingalls: When I left.

Hsu: Okay. And so then you said you were in Truckee sort of semi-retired. What time frame was that?

Ingalls: Up until I started at Sun, and I'll have to give you the years.

Hsu: I think that is 2005.

Ingalls: Yeah. That sounds right.

Hsu: So it's between 2002 and 2005.

Ingalls: Yeah.

Hsu: Okay.

Ingalls: Yeah.

Hsu: Right. So then when you went to Sun, that's when you started working on the Live Kernel?

Ingalls: Yeah. I didn't at first when I was there.

Hsu: Okay.

Ingalls: I came there, because, you know, I had worked with Glenn Edens and his crew and they had then moved to Sun. Glenn was then running research at Sun. So it sounded like fun. But I got there and I was thinking sort of, "What am I doing here in the cathedral of Java?"

Hsu: <laughs>

Ingalls: And so I can remember that the only Java program I ever wrote was that I wrote a Java interpreter for Squeak.

<laughter>

Ingalls: And I was going to release it and they said, "You can't call it J-Squeak because of the J," so we called it Potato and…

Hsu: <laughs>

Ingalls: And that sort of sat around for a while, but the reason it's significant is that it was easy enough to do that later on we converted it to JavaScript and then we had a Squeak that could run on the browser, and that became a part of the ongoing work with Squeaks that run on a browser.

Hsu: Mm-hm.

Ingalls: But that fall it seemed like, the people who were having fun there were doing web programming, and I thought, "Oh, that'd be nice to get into," and so I started looking at it and it seemed like you were having to learn CSS and PHP and JavaScript and all this other stuff and it just looked complicated. I thought, "Why can't you just do what we did in good old Smalltalk systems?" which is if you've got a dynamic language and graphics, put them together in a system like Squeak. So that Christmas I put together a system which was an implementation of Morphic in JavaScript that ran on a browser using—at

first used just the Java 2D graphics package, and we got that running and it, you know, it took a month to do, and then we worked on that some more and for a while it needed this Java 2D plugin to run. But it was right around that time when SVG Graphics was getting adopted by all the major browsers and it had all that we needed, so we realized that if we converted to using SVG Graphics, then we wouldn't need a plugin, and we were, you know, making progress with the Morphic system in JavaScript, so by the fall of that year, we released the Lively Kernel as the first version.

Hsu: Okay. Right. That was—see, what year was that?

Ingalls: It's probably 2006, probably.

Hsu: 2006.

Ingalls: In probably September or November, September, October.

Hsu: Okay. So that's actually pretty soon after you got there?

Ingalls: Yeah.

Hsu: About a year.

Ingalls: Yeah, yep.

Hsu: Right. Yeah. Huh.

Ingalls: We may have skipped a year there. I'll have to—I can line up years with you later.

Hsu: Okay. <laughs> Sure. And so you continued working on Live Kernel after that?

Ingalls: Yeah, so we've continued doing that for quite a while.

Hsu: Mm-hm.

Ingalls: Yeah. Basically doing more work to support web development and there's a lot of support in there now for doing shared access to web pages.

Hsu: Sure.

Ingalls: And collaborative support for work.

Hsu: Mm-hm. Right. And so then you joined SAP in 2010?

Ingalls: That's right.

Hsu: Yeah.

Ingalls: Yep.

Hsu: What prompted that change?

Ingalls: Well, this was about when Sun was about to be acquired.

Hsu: Right. <laughs>

Ingalls: –by somebody.

Hsu: <laughs>

Ingalls: And I didn't like all the choices, and I was talking to Alan at the time and Alan had been consulting with, ah, the Chief Technical Officer at SAP, Vishal Sikka, and he said, "Well, you ought to talk to Vishal," so I did, and it sounded like there was a place for me in [a little research group] they had there. So I did that. I joined there, and started working in a group run by Ike Nassi.

Hsu: Oh, okay.

Ingalls: So yeah.

Hsu: Okay, yeah.

Ingalls: Yep. Another old connection.

Hsu: Yeah, mutual friend of ours. <laughs>

Ingalls: Yeah, good. Yeah.

Hsu: Exactly. <laughs>

Ingalls: Yeah. So that was fun, and we started using Lively Kernel for various things that might or might not be useful in SAP and raising the consciousness about different approaches to web programming.

Hsu: Hm. Hm. And now you're working for Y Combinator.

Ingalls: That's right.

Hsu: That's a very recent development.

Ingalls: Yeah. Yeah.

Hsu: How did that happen?

Ingalls: So we had done, we had spun a group off from SAP although we were still under SAP with an office up in San Francisco. To include Bret Victor and Vi Hart, and then the opportunity came along to spin that out under, with, different financing, and so that's what we did and that's how we wound up under Y Combinator.

Hsu: Hm, okay.

Ingalls: So there's a small part of Y Combinator called YC Research.

Hsu: Ooh.

Ingalls: And that's what we're a part of.

Hsu: Okay.

Ingalls: Yeah.

Hsu: So it's really a spin-out of the SAP work?

Ingalls: Yeah.

Hsu: Okay.

Ingalls: Yeah.

Hsu: Right. Hm. And I found it kind of interesting that you've continued to do, you know, work on Squeak and Squeak-like things while not working directly with Alan, whereas Ted Kaehler, for instance, has continued to work directly with Alan at Viewpoints Research.

Ingalls: Right. Mm-hm.

Hsu: What prompted that decision to…

Ingalls: Well, so I don't live in Los Angeles right now.

Hsu: Right.

Ingalls: I may end up being there and working with Alan that way. But I think it's the, it was the geographical separation, and yes. I continue to work on Squeak-like things, but the way in which they're similar are just that they are dynamic environments that are easy to use.

Hsu: Right. <laughs>

Ingalls: And so that's what I do now, and in JavaScript, which might not be my favorite language, but there are lots of people who use it. Lively Kernel is a system that's very much like Squeak in its

malleability, and yet it's very easy to embed any other web content in. In the Smalltalk world, we were a little bit of a closed garden.

Hsu: Oh, yeah.

Ingalls: And in Lively, you can do all the, you know, interact with anything that's on the web and embed other web software in things.

Hsu: Right.

Ingalls: And yeah, and what I'm working on there right now is sort of, where I'm headed for, is to try to have an environment that kind of teaches what I would call computational thinking. In other words, what is it like to use a live object system for everything? For taking your notes, for simulating things, for giving presentations. How could those all be the same environment? And, you know, being able to add pieces of the web into it or share things, you know, be dynamically connected with people that way. So that's what we're working on right now.

Hsu: Hm.

Ingalls: And then some of the other people in the group are doing stuff with 3D and VR and so it turns out that Lively Kernel is a nice, small piece that you can put into any environment like that to give some live coding capability. So we may do some collaborations there too.

Hsu: Wow. Hm. Well, that's fascinating. <laughs> So then I take it that the—so after joining Sun and doing Lively Kernel, you ceased to work on Squeak itself?

Ingalls: Yes, that's pretty much true. Yep.

Hsu: Okay.

Ingalls: Yep. I still love it.

<laughter>

Ingalls: And I'm still interested in what people are doing with it.

Hsu: Right.

Ingalls: But yeah.

Hsu: <laughs> Okay. Well, I guess that's the end of the historical questions. I'll ask a few more sort of broader looking back questions. Let's see. Let's see, we've already talked about the restorations. Actually, maybe we could talk about that a little bit more, the various restorations of Smalltalk that you've done. So we talked about the one of Smalltalk-78. There was—several 72 ones that you've done. Sort of what motivates you to do, keep doing, these restorations?

Ingalls: Oh, okay. So restorations are fun to do, if they're interesting systems, and I consider Smalltalk-72 to be a really interesting system. So I've done a restoration of it in Squeak and then I've done one in, I've done two in Lively. One of the ones in Lively I actually simulate the Data General Nova instruction set, so it's like emulating the Alto in—to run an actual old saved image of Smalltalk-72.

Hsu: Oh, wow.

Ingalls: So that's like an Alto restoration.

Hsu: Right.

Ingalls: And–

Hsu: So you can just take the exact, an image of something that, from that era, and run it on this?

Ingalls: Yes.

Hsu: Wow.

Ingalls: Yeah. And it runs faster than it used to.

<laughter>

Ingalls: So that's pretty funny. The other one that I was doing is completely a linguistic one, which is doing just a JavaScript interpreter for the sort of theoretical form of Smalltalk-72, because I thought that I could've done a better job than I did back then. But that one's incomplete.

<laughter>

Ingalls: And then I alluded to Potato a while ago, which was my experiment to learn Java by writing a Squeak virtual machine. So the experience from that, combined with the experience of the Smalltalk-72 running so fast when double emulated, made Bert Freudenberg and I think that we ought to be able to get Squeak to run fairly simply just by transcribing my Potato, Java version, into JavaScript. So just because all the code was there, it shouldn't take him long to pick it up. So he did that, and did some great work with the storage management, which was the hardest thing to get right, and he did what's now called SqueakJS, which is a system that will run any current Squeak image or an old one in the browser, and then, based on that, there was, we had, Alan and a couple of us were interested in resurrecting the Notetaker Smalltalk, which we call Smalltalk-78. It was one of the in-between Smalltalks, and so Bert and I launched into that and got it running in a couple of weeks and, you know, then there were a couple more months of tweaking it. But so that's now—so that was a formerly dead Smalltalk that's now been revived.

Hsu: Right. I think I read something about that. Was, like, the code had been found in was it a dumpster or what? Like, there was some sort of—rescued it from somewhere.

Ingalls: Yes. So when our group cleared out from Xerox PARC, a number of disk packs went to the dumpster and a couple of the techs there picked up the disk packs and kept them and got them transferred onto CDs, and so those are saved, and we've managed, to resurrect some of those. The Smalltalk-72 that I mentioned that was a direct load of that file, that came off of one of those disks.

Hsu: Oh, okay.

Ingalls: And the Smalltalk-76 as well.

Hsu: Right.

Ingalls: And the Notetaker Smalltalk, yeah.

Hsu: Okay. Wow. So you got a lot of mileage out of that?

Ingalls: [Definitely]

Hsu: <laughs>

Ingalls: It was really fortunate those guys saved those, because I had written stuff for those systems but I didn't keep real files.

Hsu: Right. Wow. Hm. Okay. So how much of an impact in your view has Smalltalk made on the computer industry or computer science? <laughs>

Ingalls: Boy. Well, I think Smalltalk is pretty widely regarded as the original object-oriented system and a fairly pure version of it, and you can see its effect. I mean, the Objective-C that's used in Apple came directly [via] the guys at Schlumberger who picked up Smalltalk, and they did the conversion to adapt it to work with C. So there's a direct lineage there, and the other, you know, C++, to a lesser degree, but certainly—and then the debugging frameworks. Smalltalk's debugging framework from the beginning was, you know, everybody in the industry said, "Woah, I want something like that," and so Eclipse and other systems like that are directly modeled on it, and so I think there's been a big impact there. It's funny though, the real productivity and liveness that some of these systems have has not been picked up in the industry as much as I would expect. There are a lot of people, I mean, there are reasons for, you know, having serious software control procedures, but there are a lot of people in the industry who just don't get what a difference it makes to have things be completely live. That's somewhat separate from being object-oriented but that's something that if a system is well designed and object-oriented it can be completely live, and I think that's scary to some people or something.

Hsu: <laughs>

Ingalls: But there's still some evolution of the world based on what Smalltalk and such systems had to contribute that's yet to happen, I think, and I think it may, some of it may happen, actually, in the more radical areas, like people in VR wanting to do live coding in VR, and will get that—they don't have any history, you know, and they just want to make things happen.

Hsu: Yeah. Do you think it's, I mean, this is something I've been thinking about as well, but is there something about commercial user environments that platform vendors don't want users to be mucking around or changing things willy-nilly?

Ingalls: Oh. Well, there are reasons not to change things willy-nilly.

Hsu: <laughs>

Ingalls: Which is, you know, then the documentation, gets out of sync and this and that. But that's not a reason to not have the stuff that you're building be immediately responsive and to be able to evolve it while it's happening.

Hsu: Right.

Ingalls: I mean, it's so valuable. If you're writing some software for a client, to be able to sit down with that client and show them your first version, you know, a quarter of the way into the project, and they can look at it and they can say, "Oh, can you change this?" and you change it then and there and in that kind of a situation, you're moving forward [much] faster.

Hsu: Right.

Ingalls: And for another thing, if you're moving forward faster, then [if] it turns out the person says, "But that's not really what I want," and you [avoid doing] a whole thing that would've been a waste of time. You get into the direction that the person really wanted, and that's incomparably more productive.

Hsu: Right.

Ingalls: So…

Hsu: So you don't see it as something where the user themselves have to be able to make the changes, but more of this rapid iteration or kind of like Agile way of development is really the benefit?

Ingalls: Yeah. I mean, sometimes it is the user. I mean, it depends on whether a person is having somebody build something for them or whether they're building it themselves.

Hsu: Right.

Ingalls: Yeah.

Hsu: Yeah, uh-huh. That's basically that question. Do you feel like there's been a shift in the industry now away from dynamic object-oriented languages? It seems like a lot of the, you know, like Apple, for instance, the Swift language is a lot less, it's more static than, it's more strongly typed, than Objective-C was, and so there seems to be a shift away from things like Ruby and Python and more towards statically compiled languages again. What do you think is sort of the reason for that trend?

Ingalls: Well, let's see. There's nothing bad about types and there's nothing not dynamic about types.

Hsu: Right.

Ingalls: I think it's possible to have your cake and eat it too. I do think that as we go forward we need to be building systems that are increasingly more secure, increasingly reliable, and types are just one form of assertion that can be checked to assure that things are being done right. But I think it's possible to have dynamic type systems that are still effective that way, and I don't know about sort of trends in the industry at this point. I'm not the best person to ask about that. But I do know that being able to keep things live makes for better productivity. In my experience.

Hsu: Right. Do you have any suggestions for how to get that message out to <laughs> the rest of the industry, which doesn't seem to be aware of this? <laughs>

Ingalls: No, I don't know. I mean, there's only so much you can do by standing on a street corner and yelling.

Hsu: <laughs>

Ingalls: I mean, most of the important things that have been adopted get taken up by, you know, survival of the fittest. People look and see what works and people look and see what doesn't, and generally you can't tell somebody they should be working differently.

Hsu: Right. So that, I mean, that's a metaphor that you've used before is natural selection. So has it been, has that been true in your experience, that good technology always eventually wins in the end, in the long run?

Ingalls: Well, I think so. There are always the more conservative and the more radical processes going on and being a researcher I'm more interested in the most dynamic, the most flexible thing, and there's no doubt that in certain environments you want things to be very carefully controlled, that maybe it's a force that goes against complete malleability. But I think there are places where you want to keep things live and malleable and places where you want to keep things secure. I don't know exactly what's to be said about it.

Hsu: Mm-hm. Hm. Where do you think the industry is going in the future?

Ingalls: Well, there's a lot of industry.

Hsu: <laughs>

Ingalls: You know, it depends on whether we're talking about the programs for self-driving cars or for virtual reality games or what. Or banking or, you know. I think a lot of systems that are well understood will continue to be done in well understood software systems. I think there's a lot of progress to be made in security from where we are now. I really like, I have liked for a long time, a lot of the work that was started with the work on E, and real capabilities. I think that that makes for, more secure systems, and I think the question of how to achieve that together with the kind of liveness that you see in a system like Squeak or Lively is a good challenge, and it's one I've been interested in and I'm not pursuing it right now, but I've certainly had a bunch of good conversations with Mark Miller about it. Yeah, I don't know, and there are a bunch of new things coming on. I mean, people working in virtual reality with newer languages, but I think it's going to be interesting to see what comes out of that.

Hsu: Last but not least, what advice would you give a young person today?

Ingalls: Okay.

Hsu: Or a young person starting in computing today.

Ingalls: Right.

Hsu: Maybe to narrow it down a little bit. <laughs>

Ingalls: Yeah. Well, I guess first figure out what it is that you want to do. Why are you studying software? Why are you in computers? And then look around at it, given what you want to do, that may suggest what kind of languages you're interested in or what kind of—are you interested in graphics or music, what it is that you want to do? And I always think that to find the system that's as malleable as possible that you can most easily see doing what you want to do, and I guess nowadays it would be good to find other groups that are doing that kind of work. Talk to them about the latest thing that they're working with, because a lot of it is changing, and then start to build things. Because that's where you learn. You know, you learn by building things for yourself or for other people, and write about it. It's interesting, a lot of systems that I've improved, I've improved the most when I'm writing documentation about them, because I start to write about how to use them and I realize that I'm telling somebody how to do something that's more complicated than I want to say. So then I go back to the system and make it so I don't have to say that in the documentation.

Hsu: <laughs>

Ingalls: And that's how things get simpler and better.

Hsu: Hm. So tell us a bit about these documents that you've brought and the project that you worked on.

Ingalls: Okay. So in my last year at Harvard, I was, I had gotten interested in analog computing because it's so live, and it was interesting that the different components actually gave, you know, manifested mathematical behavior in actual voltages and then I read another article about how you could represent analog quantities similar to voltages but as pulse trains on a wire in a digital world. So the rate of arrival of pulses could represent a quantity, and I thought that that way you could actually build a thing like an analog computer but using digital components so it would be reliable and accurate, and the thought that I had, the simplest version of this, would be to have a circuit that's called a bitrate multiplier, which is one where you have a pulse train coming in and then you have a register that holds a value and the logic in the bitrate multiplier causes the output pulse train to have a rate of pulses that's proportional to the quantity in the register, and then I thought that you could use that and feed it back to itself to either increase, if this register were a counter, either increase or decrease the value, that would give you an exponential decay or an exponential increase of the digital value. This was all at the time that Fairchild had just come out with these little integrated circuits that were a dollar apiece, that somebody of my capability could wire together. So it seemed possible to me to actually build a computer that would do exponential decays and then by that you could actually compute logarithms by tracking the time to decay to a certain value. So if you could calculate logarithms then you could actually accumulate the logarithms and do multiplication. So you had the makings there of a complete calculator. So I set about to build this and here's a picture of the finished product and that's the most complicated thing I ever put together, and I'll show you the—there's the back side of the wiring, all of which–

Hsu: Raise it up a little bit.

Ingalls: Okay.

<crew talk>

Ingalls: A lot of connections that had to not fail.

Hsu: <laughs>

Ingalls: And anyway, this thing succeeded in taking logarithms and you could, it would accumulate the logarithms, and then you could wire it back to count up and it would do exponential so that you could multiply and divide and you could do it in a similar mode to add and subtract, and the especially cool thing about the logarithms, about doing calculation this way, was—I don't know if you've ever worked with a

slide rule. Well, a slide rule has the problem that when you get a result that's off the end of the slide rule, you have to go backwards and divide everything by 10. But if you have a circular slide rule, you always, get the significant digits back of your result. Well, I made it possible, so that overflow in this register corresponded exactly to a factor of 10 so that you could keep multiplying numbers and it would not lose precision off the numbers but would just take you to the next decade of result. So it had the effect of being a circular slide rule as well.

Hsu: <laughs>

Ingalls: Anyway, so that's the most complicated computer I ever actually built, and I think it was as a result of that that I thought that working in software would be so <laughs> much simpler.

Hsu: <laughs>

Ingalls: And that's, that's how I've been ever since.

Hsu: I see. <laughs> So it's fascinating. So you mentioned that you liked analog computing because it was live, so this idea of liveness or instant interactivity or I guess in another paper you call it the reactive principle?

Ingalls: Mm-hm.

Hsu: Has sort of been with you from quite early on.

Ingalls: Yeah, from the very beginning. Yeah. And that's something, I don't know how you teach that, except to give people a system that behaves that way, and once you've had that feel, you know, you never want to go back and have to wait for things to compile and load.

Hsu: Right. It's that instant gratification that you've just talked about.

Ingalls: Yeah, yeah, and it isn't just instant gratification. It's—the world is that way. If you go out to a steam plant and turn a wheel, you know, it does what it does [then and there], and the things that we build with software should be at least that lively.

END OF THE INTERVIEW

